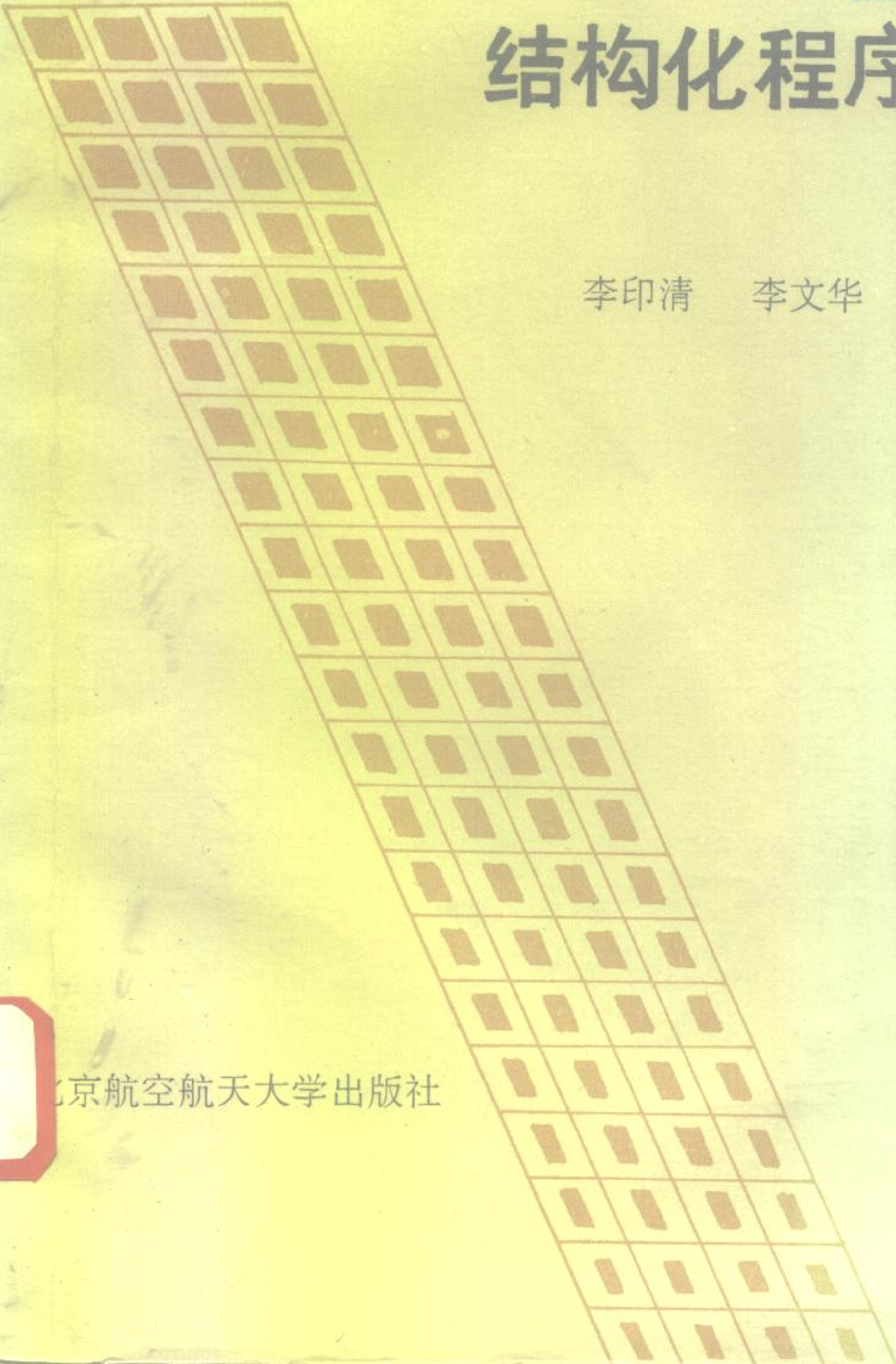
TRUE BASIC
结构化程序设计

李印清　李文华　编著



北京航空航天大学出版社

True BASIC 结构化程序设计

李印清　李文华　编著

北京航空航天大学出版社

(京)新登字166号

内 容 简 介

本书全面、系统地介绍了True BASIC(2.03版)的各种语言成分，着重讲述了结构化程序设计以及算法设计的基本方法和策略。全书共十二章和两个附录。第一章介绍了算法的概念、表示，算法设计的基本方法和策略，以及富有生命力的一种软件设计表现法——PAD。第二至十一章介绍了True BASIC程序设计的主要内容，如输入、输出、处理、数组、函数、子程序、库和模块，并结合结构化程序设计思想，讲述程序的模块化设计、菜单驱动程序技术、递归技术、文件处理、作图技术、调试和陷井技术等高级程序设计的概念和技巧。第十二章详尽地介绍了True BASIC程序的上机操作。书末给出了两个附录，以供读者查阅。

本书可作为高等院校非计算机专业计算机学习的基础教材或教学参考书，同时可作为计算机培训班的教材，还可供从事经济工作、企事业管理及其他有关人员自学和参考。

书　　名：True BASIC结构化程序设计
True BASIC JIEGOUHUA CHENGXUSHEJI
编　　著：李印清　李文华
责任编辑：娄铁军　齐桂森
出　　版：北京航空航天大学出版社(北京市学院路37号，100083)
发　　行：新华书店总店科技发行所
销　　售：本社及各地新华书店
印　　刷：朝阳科普印刷厂
开　　本：787×1092　1/16
印　　张：18.75
字　　数：480千字
印制日期：1994年10月第1次印刷
印　　数：13000册
书　　号：ISBN 7-81012-509-5/TP·126
定　　价：12.95元

前　言

True BASIC(2.03 版本)是 BASIC 创始人 John G. Kemeny 和 Thomas E. Kurtz 于 1987 年推出的最新 BASIC 版本。它保留了 BASIC 语言的人机交互、易学易用的基本特点，并对 BASIC 和 True BASIC 1.0 版本作了大量修改，同时吸收 FORTRAN、PASCAL 等语言的长处，使之成为一种完全结构化的程序设计语言，为用户提供了一整套按照软件工程化要求开发软件的重要手段。

True BASIC(2.03)的主要特点是：

(1) 典型的结构化程序设计语言

这使得设计的程序结构清晰，易阅读，易维护。在程序的编制过程中，如果大量使用 BASIC 的 GOTO、GOSUB、ON-GOTO 等语句，势必使程序的逻辑流程紊乱，结构缠绕不清。True BASIC 从根本上改变了这种结构，抛弃了行号，废除了 GOTO 等语句，充实了选择结构和循环结构。另外，引入了 EXIT 簇语句；以解决结构化程序有时执行效率低下的问题。

(2) 是一种“两栖型”语言

True BASIC 是一种既有解释器又有编译器的“两栖型”语言。在编写、调试程序时使用它的解释器，此时用户处于全屏幕编辑状态，可以对指定语句行或标定的行模块——多行组、一个子程序、一个函数、一个图画进行编辑、复制、删除、移动等操作，使查错、纠错简单易行，提高编写、调试程序的效率。程序一旦调试通过，则用它的编译器将程序优化编译，并经汇集以生成执行文件，日后使用时可只用它的可执行文件。

(3) 功能丰富，有足够的能力处理各种较大的应用课题

与任何一种高级语言相比，在语言提供的功能方面 True BASIC 毫不逊色。它具有极其丰富的各种功能，如字符串处理功能；矩阵运算功能；文件处理功能；高效率的作图、动画功能；库文件和外部函数、子程序和图画的递归调用功能；音响功能；多窗口操作功能；以及时钟、日期功能；菜单功能；以绝对地址方式读写内存功能等等。

True BASIC(2.03)还提供了与 DOS 操作系统的简练而方便的接口方式，可在程序中直接调用 DOS 的命令和功能，并可与外部的可执行文件进行链接。

(4) 良好的通用性

True BASIC 是严格按照美国国家 BASIC 标准编写的，具有良好的可移植性。与 BASIC 有较强的兼容性，在 True BASIC(2.03)中还提供了自动变换程序。

为了适应计算机科学技术的迅速发展，满足计算机基础教学的需要，我们在总结多年教学实践的基础上编写了此书。

全书共十二章。第一章可作为全书的导引，讲述算法的概念、表示，算法设计的基本方法和策略，以及富有生命力的一种软件设计表现法——PAD。第二至十一章全面系统地介绍了 True BASIC 程序设计的主要内容，如输入、输出、处理、选择、循环、数组、函数、子程序、库和模

块，并结合结构化程序设计思想，讲述程序的模块化设计、菜单驱动程序技术、递归技术、文件处理、作图技术、调试和陷井技术等高级程序设计的概念和技巧。第十二章详尽地介绍了 True BASIC 程序的上机操作。书末给出了两个附录，以供读者查阅。

“授之一鱼，莫如授之以渔。”本书注重基本算法、结构化程序设计方法的讲述和训练。书中不是简单地罗列语法规定和使用细节，而是在分析问题的基础上着重于被处理对象（数据）的抽象以及从计算机的操作的角度对解题过程的抽象。本书的另一特点是利用先进的图形工具 PAD 来描述算法，使算法执行逻辑透明可视。

本书编写时力求做到由浅入深、通俗易懂，以及知识结构的逻辑性。在此基础上列举了大量例题，以帮助读者掌握算法设计和编写程序的方法，不少例题取材新颖且具有实用价值。书中各章习题数量较多，难易程度合理搭配，为教学提供了充分的选择余地。

本书可作为高等院校非计算机专业计算机学习的基础教材或教学参考书，讲授学时为 50～70。在学时少的情况下，可根据学生的情况酌情删去书中某些内容，本书目录页中给出了参考性意见（即建议删去带**的章节）。本书同时可作为计算机培训班的教材，还可供从事经济工作、企事业管理及其他有关人员自学和参考。

本书编写过程中，得到许多同志的热情帮助。王保正、罗道昆、赵逢余等同志为全书的编写提出许多宝贵的意见，在手稿处理过程中还得到牛晓太、吴昊等同志的不少帮助。藉此机会谨向他们以及本书中所借鉴、引用的参考文献的所有作者一并表示衷心的感谢。

本书第 1、6、7、8、10、12 章由李印清编写，第 2、3、4、5、9、11 章由李文华编写，李印清拟定了该书的编写大纲，编撰了两个附录，并对全部书稿进行了统修。

由于时间仓促，学识和水平有限，书中谬误之处在所难免，敬请读者批评指正。

编著者
1994 年 5 月

目　录

第一章　算法设计与PAD

第一节　算法……（1）
一、算法的概念……（1）
二、算法的特征……（3）
三、算法的表示……（4）
第二节　设计算法的基本方法和策略……（6）
一、迭代法……（7）
二、枚举法……（7）
三、递推法……（8）
四、递归法……（9）
五、分治法……（9）
* * 六、回溯法……（9）
第三节　PAD的基本原理与图式
一、PAD的基本原理……（11）
二、PAD的基本图式……（11）
三、PAD的画法……（13）
第四节　基于PAD的编程方法……（15）
习题……（18）

第二章　True BASIC基本知识

第一节　True BASIC简介……（20）
第二节　True BASIC的程序格式……（20）
第三节　True BASIC字符集、常量、变量……（21）
一、字符集……（21）
二、常量……（21）
三、变量……（22）
第四节　标准函数……（23）
一、常用数学函数……（23）
二、三角函数……（24）
三、字符串函数……（25）
第五节　表达式……（26）
一、数值表达式……（26）
二、字符串表达式……（27）
第六节　True BASIC上机操作初步……（28）

一、True BASIC 的启动与退出 …… (28)
二、True BASIC 屏幕窗口 …… (28)
三、源程序的编辑 …… (29)
四、解释，编译，汇集 …… (30)
五、程序文件的保存与调用 …… (30)
习题 …… (31)

第三章　数据的输入与输出

第一节　PRINT（屏幕输出）语句 …… (33)
一、输出内容 …… (33)
二、输出格式 …… (34)
第二节　LET（赋值）语句 …… (36)
第三节　INPUT（键盘输入）语句 …… (38)
第四节　READ/DATA（读/置数）语句与 RESTORE 语句 …… (40)
一、READ/DATA 语句 …… (40)
二、MORE DATA 与 END DATA 函数 …… (42)
三、RESTORE 语句 …… (43)
第五节　REM（注释）语句和 END（结束）语句 …… (44)
一、REM 语句 …… (44)
二、END 语句 …… (45)
第六节　程序设计初步 …… (45)
习题 …… (46)

第四章　选择结构

第一节　关系表达式和逻辑表达式 …… (49)
一、关系运算符与关系表达式 …… (49)
二、逻辑运算符与逻辑表达式 …… (50)
第二节　IF 型选择结构 …… (51)
一、简单 IF 选择语句 …… (52)
二、IF-THEN-ELSE（二择一）语句 …… (53)
三、ELSE IF（多择一）语句 …… (54)
第三节　SELECT CASE 型选择结构 …… (56)
第四节　选择结构应用举例 …… (58)
习题 …… (62)

第五章　循环结构

第一节　FOR 循环和 EXIT FOR 语句 …… (66)
一、FOR 循环 …… (66)
二、EXIT FOR 语句 …… (73)
第二节　DO 循环和 EXIT DO 语句 …… (74)
一、DO 循环 …… (74)

二、EXIT DO 语句 …… (78)
第三节　循环嵌套 …… (79)
第四节　循环结构应用举例 …… (83)
一、枚举算法设计 …… (84)
二、迭代算法设计 …… (86)
习题 …… (88)

第六章　数组与 MAT 簇语句

第一节　数组及数组元素 …… (93)
一、数组的概念 …… (93)
二、数组元素的表示方法及数组的维数 …… (94)
三、数组的存储结构 …… (94)
第二节　DIM(数组定义)语句和 MAT REDIM(数组重定义)语句 …… (95)
一、DIM 语句 …… (95)
二、MAT REDIM 语句 …… (99)
三、数组测试函数 …… (100)
第三节　数组的整体输入与输出 …… (100)
一、MAT READ 语句 …… (100)
二、MAT PRINT 语句 …… (102)
三、MAT INPUT 和 MAT LINE INPUT 语句 …… (103)
第四节　数组赋值与运算 …… (104)
一、数组赋值语句 …… (104)
二、数值量与数组相乘 …… (105)
三、数组的加、减运算 …… (106)
四、两矩阵相乘 …… (106)
五、内部数组与矩阵函数 …… (107)
第五节　排序 …… (108)
一、选择排序 …… (108)
二、交换排序 …… (110)
第六节　应用程序举例 …… (112)
一、数组的计算与统计 …… (112)
二、查找 …… (113)
* * 三、数组的插入与删除 …… (117)
习题 …… (119)

第七章　自定义函数、子程序、库和模块

第一节　自定义函数 …… (123)
一、内部函数 …… (123)
二、外部函数 …… (130)
* * 三、递归函数程序设计 …… (132)
第二节　子程序 …… (134)

一、内部子程序 …………………………………………………… (135)
二、外部子程序 …………………………………………………… (140)
* *三、递归子程序设计 …………………………………………… (145)
* *四、回溯程序设计 ……………………………………………… (147)
第三节　程序的链接…………………………………………… (150)
一、CHAIN(链接)语句 …………………………………………… (150)
二、PROGRAM(程序)语句 ……………………………………… (150)
第四节　库…………………………………………………… (151)
一、如何建立库 …………………………………………………… (151)
二、如何使用库 …………………………………………………… (152)
* *第五节　模块…………………………………………………… (153)
一、如何定义模块 ………………………………………………… (153)
二、如何使用模块 ………………………………………………… (156)
习题…………………………………………………………… (157)

第八章　格式化输入与输出

第一节　格式化输出…………………………………………… (160)
一、行宽和区宽的设置 …………………………………………… (160)
二、TAB 函数 ……………………………………………………… (161)
三、PRINT USING(自选格式输出)语句…………………………… (162)
四、MAT PRINT USING 语句 …………………………………… (166)
第二节　屏幕的定位输入……………………………………… (169)
一、屏幕的光标定位控制 ………………………………………… (170)
二、GET KEY(单键输入)语句 …………………………………… (171)
三、KEY INPUT(输入测试)函数………………………………… (173)
第三节　菜单驱动程序设计…………………………………… (173)
一、根据功能名称或数字选取菜单 ……………………………… (174)
二、下拉式菜单驱动程序的设计 ………………………………… (174)
第四节　菜单驱动程序设计举例……………………………… (179)
习题…………………………………………………………… (189)

第九章　磁盘数据文件

第一节　文件基本操作………………………………………… (190)
一、通道与文件指针 ……………………………………………… (190)
二、OPEN(打开文件)语句 ……………………………………… (190)
三、CLOSE(关闭文件)语句 ……………………………………… (192)
四、文件逻辑函数与设置文件指针语句 ………………………… (192)
五、文件删除语句 ………………………………………………… (193)
六、ASK(文件查询)语句 ………………………………………… (193)
第二节　正文文件……………………………………………… (194)
一、正文文件的概念与特点 ……………………………………… (194)

二、PRINT #（写正文文件）语句……(195)
三、INPUT #（读正文文件）语句……(196)
四、设置行宽，域宽语句……(199)
五、向打印机输出数据……(200)
六、正文文件应用举例……(201)
* *七、显示格式文件……(205)
第三节　记录文件……(209)
一、记录文件的概念与特点……(209)
二、WRITE #（写记录文件）语句……(210)
三、READ #（读记录文件）语句……(211)
四、设置记录文件指针语句……(212)
五、具有多数据项的记录文件……(213)
六、记录文件应用举例……(215)
* *第四节　字节文件……(224)
一、WRITE #（写字节文件）语句……(224)
二、READ #（读字节文件）语句……(224)
三、字节文件应用举例……(224)
习题……(227)

第十章　图形程序设计

第一节　作图环境……(229)
一、屏幕模式……(229)
二、设置图形窗口……(230)
三、图形着色……(231)
第二节　基本作图语句……(233)
一、PLOT POINTS（画点）语句……(233)
二、PLOT LINES（画线）语句……(233)
三、PLOT AREA（区域着色）语句……(234)
四、画矩形和实心矩形……(235)
五、画圆和实心圆……(235)
六、图形中的正文设置……(236)
七、MAT PLOT 簇语句……(237)
* *第三节　动画和图画结构……(238)
一、动画……(238)
二、图画结构……(239)
* *第四节　图形输入和多窗口操作……(240)
一、GET POINT（图形输入）语句……(240)
二、多窗口操作……(240)
习题……(242)

第十一章　程序的调试与出错处理

第一节　程序中的常见错误类型及调试……(243)

一、程序中的常见错误类型 …………………………………………………………………… (243)
二、程序调试技术 …………………………………………………………………………… (244)
**第二节　出错处理结构…………………………………………………………………… (246)
**第三节　与出错处理有关的函数和语句………………………………………………… (247)
一、出错函数 ………………………………………………………………………………… (247)
二、EXIT HANDLER(退出错误处理模块)语句 …………………………………………… (249)
三、CAUSE ERROR(捕错)语句 …………………………………………………………… (251)
习题……………………………………………………………………………………………… (252)

第十二章　True BASIC 上机操作指导

第一节　True BASIC 的运行环境 ………………………………………………………… (253)
第二节　全屏幕编辑………………………………………………………………………… (253)
一、光标的移动 ……………………………………………………………………………… (253)
二、初级编辑功能键 ………………………………………………………………………… (254)
三、高级编辑功能键 ………………………………………………………………………… (254)
四、True BASIC(2.03 版)新增的编辑功能键 …………………………………………… (255)
第三节　True BASIC 系统命令 …………………………………………………………… (256)
一、屏幕设置命令 …………………………………………………………………………… (256)
二、文件处理命令 …………………………………………………………………………… (256)
三、程序编辑命令 …………………………………………………………………………… (259)
四、程序调试命令 …………………………………………………………………………… (261)
五、程序的编译、汇集和运行命令…………………………………………………………… (262)
六、其他命令 ………………………………………………………………………………… (264)
第四节　True BASIC 与 DOS 系统以及.EXE 文件间的接口 …………………………… (265)
第五节　在 CCDOS 系统下使用 True BASIC 的方法 …………………………………… (267)
附录 A　IBM-PC 字符与 ASCII 代码对照表……………………………………………… (268)
附录 B　出错信息注释……………………………………………………………………… (273)
参考文献……………………………………………………………………………………… (288)

第一章　算法设计与PAD

要设计出好的程序，必须先设计出好的算法。算法是精确定义的一系列规则，指出怎样从给定的输入信息经过有限步骤产生所求的输出信息。

在这一章中，首先讲述算法的概念、特征，流程图表示以及设计算法的基本方法和策略，然后介绍极富生命力的软件设计表现法——PAD，以及基于PAD的编程方法。

本章可以作为全书的导引，其中介绍的有些概念和方法可能在后续章节或者读完全书后才会有深刻的理解。

第一节　算　法

一、算法的概念

什么是算法？著名计算机科学家D. E. Knuth在《THE ART OF COMPUTER PROGRAMMING》一书中称："一个算法，就是一个有穷规则的集合，其中之规则规定了一个解决某一特定类型的问题的运算序列。"

实际上，设计算法并非神秘莫测，它与人们做任何事情相仿，均要在规定的解题环境中，采用所允许的基本操作去构造解题的步骤。注意，这里所说的"解题"泛指解决某一类问题，而不仅仅是指"计算"。

设计由计算机执行的算法，必然涉及计算机的功能，即它能做什么。尽管我们今天可以见到种类繁多、大小各异的计算机，但它们都具备以下最基本的操作功能：

算述运算：加、减、乘、除。

逻辑运算："与"、"或"、"非"。

比较运算：>、<、=、≠。

数据传送：输入、输出、赋值。

控制转移。

虽然上述运算和操作在不同系列的计算机上的表现形式不尽相同，但在设计算法时均应按照解题要求，从上述基本操作中选择合适的操作构成解题的操作序列。

下面通过几个简单问题说明设计算法的思维方法。

〔**例1-1**〕　有两个变量a和b，要求将它们的值互换。

算法分析：由于计算机存取信息的特征是：将一个数据或某变量的值自某个存储单元向另一存储单元传送时，该单元中的原有数据将由新值替代，而发送项存储单元的值保持不变。因此不能直接做a、b的值互相传递的操作，需要引入一个临时变量t(t对应于一个存储单元)。算法可设计为：

```
s1：输入a、b的值；
s2：作a⇒t；
```

```
s3:作 b⇒a;
s4:作 t⇒b;
s5:输出 a、b 的值;
s6:结束。
```

其中,s1,s2,……分别表示操作序列的第一步,第二步,……。“a⇒t”表示赋值传送,即将 a 的值传到 t 所对应的存储单元中。

〔**例 1-2**〕 计算函数 M(x)的值。函数 M(x)为:

$$M(x) = \begin{cases} bx/h & x \leqslant a \\ a(h - x)/h & x > a \end{cases}$$

其中,a,b,h 为常数。

算法分析:首先引入变量 M,它代表要计算的函数值。根据计算机具有逻辑判断的基本功能,算法可书写为:

```
s1:输入 a,b,h,x 的值;
s2:判断 x≤a? 若条件成立执行 s2.1,否则执行 s2.2;
  s2.1:作 bx/h⇒M;
  s2.2:作 a(h-x)/h⇒M;
s3:输出 M 的值;
s4:结束。
```

算法中利用了计算机具有逻辑判断的功能,即判断“x≤a”,若真(true),执行“bx/h⇒M”,然后执行“输出 M 的值”;若假(false),执行“a(h-x)/h⇒M”,然后执行“输出 M 的值”。

〔**例 1-3**〕 计算 Fibonacci 数列:1,1,2,3,5,8,13,21…前 n 项的值。

算法分析:从数列的第 3 项开始,每项都是其前两项之和,即 $F_i = F_{i-2} + F_{i-1}$(i=3,4,5,6,…)。算法可设计为:

```
s1:输入 n 值;
s2:作 1⇒f1,1⇒f2;
s3:输出 f1,f2 的值;
s4:作 3⇒i;
s5:判断 i≤n? 若条件成立,执行 s5.1~s5.5 规定的操作;否则执行 s7。
  s5.1:作 f1+f2⇒f;
  s5.2:输出 f 的值;
  s5.3:作 f2⇒f1;
  s5.4:作 f⇒f2;
  s5.5:作 i+1⇒i;
s6:转向 s5;
s7:结束。
```

需要说明的是,算法中的 s5 和 s6 构成了一个循环处理过程。“f1+f2⇒f”表示计算下一项的值,“f2⇒f1”和“f⇒f2”操作是为下一次重复处理作准备工作。也就是说,每次重复处理时,f1 和 f2 已经获得新值。

进一步分析上述算法,可以发现其中的 s5、s6、s7 三个操作步骤的执行顺序可改写为:

```
s5：当 $i \leqslant n$ 时，重复执行下面的操作：
    s5.1：作 $f1+f2 \Rightarrow f$；
    ⋮
    s5.5：作 $i+1 \Rightarrow i$；
s6：结束。
```

由上述例题可以看出，一个算法由若干操作步骤构成，并且这些操作是按一定的控制结构所规定的次序执行。如例 1-1 中的各操作是顺序地执行的，称之为顺序结构。例 1-2 中的 s2 又包含 s2.1 和 s2.2 两个操作，它们不能同时被执行，需要根据条件成立与否决定执行哪个操作，这种结构称为选择结构。例 1-3 中的 s5（指改写后的）中的 s5.1～s5.5 是重复执行的，并且一直延续到条件“$i \leqslant n$”不成立为止，这种结构称为重复结构，亦称为循环结构。

人们已经证明：任何复杂的算法都可以用顺序、选择、重复三种控制结构组合而成。这三种控制结构称为算法的三种基本控制结构。

说读者而言，理解算法由一步一步的操作和控制结构两个基本要素组成是必要的。设计算法与演绎数学有明显区别，后者以公理系统为基础，通过有限次推演完成问题的求解，每次推演都是对问题作更进一步的描述，如此不断推演，直到能将问题的解描述出来为止。而设计算法则是利用解题环境提供的基本操作对输入数据进行逐步加工、变换和处理，从而实现解题目标的。

二、算法的特征

依据 D. E. Knuth 的观点，算法应具有以下几个特征：

1. 确定性

算法的每一步其含义必须有确切的定义，即每一步运算应该执行何种操作必须是明确无误的，不应该存在歧义性，使执行者无所适从。

2. 有效性

算法中的每一步操作都应该是能有效地执行的，如果算法中有一个操作是不可执行的，整个算法就不具有有效性。例如，出现“计算 s/0”就不能进行有效的操作。

3. 有零个或多个输入

所谓算法的输入是指在算法开始之前对算法最初给定的量。这些输入取自于特定的对象集合。例如，例 1-1 算法中有 2 个输入，即需要输入 a 和 b 两个值。

4. 有一个或多个输出

所谓输出是指与输入有某种特定关系的量。例 1-2 算法的输出有一个，即 M 值。没有输出的算法是没有意义的。例如，求方程 $ax^2+bx+c=0$ 的实根，若任输入一组值，方程可能有解，也可能无解，无解时算法应给出无解信息。

5. 有穷性

一个算法应当包括有限个操作步骤，而不应该是无限的。换言之，一个算法必须总是在执行了有穷步的运算之后终止。

应当说明，就算法的应用性而言，有穷性的限制是不强的。一个有实用价值的算法应该不仅要求有穷的步骤，而且应该是很有限的步骤。例如，求线性方程组的解，理论上可以用行列式解法，但是要解 n 阶方程组，要计算 n+1 个 n 阶行列式的值，总共要做 $(n!)(n-1)(n+1)$ 次乘

法运算。设 n 为 20，采用每秒 1 千万次的计算机，要连续运算 3 千万年才能完成。显然这种算法是没有实际意义的。由此可知，对算法的效率要作分析，对不能在相当有穷步内终止的算法则不必在计算机上运行，对这类问题的求解应另辟蹊径。

凡是一个算法，都必须满足以上特征。需要说明的是，仅满足前四个特征的一组规则不能称为算法，而把它称作计算过程。操作系统就是计算过程的一个重要例子。设计操作系统的目的是为了控制各种作业的运行，当没有作业时，这一计算过程并不终止，而是处于等待状态，一直等到一个新的作业进入。

把一个算法视为一个函数，对深入理解算法概念会有所帮助。数学中，如果一个函数定义为：$f(x)=x^2$，则 $-\infty<x<+\infty$ 是 $f(x)$ 的定义域，$0\leqslant f(x)<+\infty$ 是其值域。类似地，把一个算法视为一个函数，是因为它把输入值的集合变换为输出值的集合，算法的定义域是一切使该算法有定义的输入值的集合，算法的值域是算法输出的一切可能值的集合。可见，一个算法定义了一个函数或一个映射，它把输入值变换为输出值。算法的输入和输出值可以是任何可表示的符号，可以是数、字母字符、特殊字符或任何其他有意义的能用计算机表示的符号。

三、算法的表示

语言是思想的外壳，设计的算法也要用“语言”恰当地表示出来。通常，可以使用以下三种类型的工具描述算法。

1. 自然语言

自然语言即人们使用的语言，如汉语、英语等。用自然语言描述算法通俗易懂，但它存在着难以克服的缺陷。

(1) 易产生歧义性。自然语言往往要根据上下文才能判别其含义，不太严格。

(2) 语句比较繁琐冗长，并且很难清楚地表达算法的逻辑流程。如果算法中包含判断，循环处理，尤其是这些处理的嵌套层数增多，用自然语言描述其流程既不直观又很难表达清楚。

(3) 当今的计算机尚不能处理用自然语言表示的算法。

2. 专用工具

为了形象地描述算法，人们创造了许多专用工具来描述算法。常用的有流程图，PAD 图，N-S 图等。除图形工具之外，人们可使用代码符号（如伪代码）描述算法。

3. 程序设计语言

程序设计语言是可以让计算机接受、辨认和执行的算法描述工具。任何算法最终都要用程序设计语言描述出来，才能在计算机上实现算法欲解决的问题。

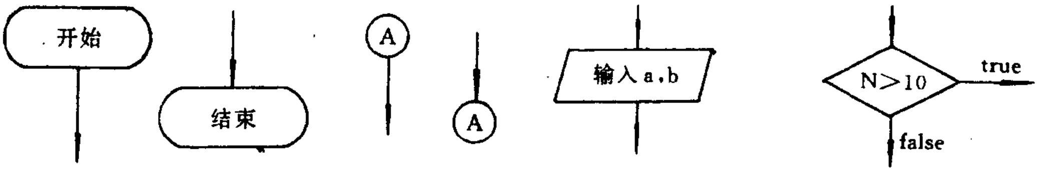
由于用程序设计语言描述的算法其表现形式是串行的，难以直观地反映算法的逻辑结构，因此，在算法设计阶段人们仍然需要利用图形工具描述算法，然后再利用程序设计语言将算法翻译成可以由计算机处理的语句序列，即程序。这一工作过程被称之为编码。

下面简要介绍流程图的使用方法。

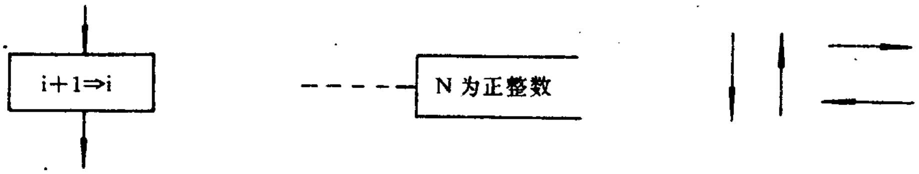
流程图是采用不同的几何图形来描述算法的逻辑结构，每个几何图形表示不同性质的操作。图 1-1 是 ANSI（美国国家标准化协会）规定的一些常用流程图符号。

起止框、连接框：表示算法的起始、终止和连接，框内填写文字或字母，如图 1-(a)所示。

输入输出框：表示输入数据和输出计算结果，框内应填写需要输入或输出的量。如图 1-1 (b)所示。


开始
结束
A
A
输入 a,b
N>10
true
false

(a) 起止框、连接框　　(b) 输入输出框　　(c) 判断框



(d) 处理框　　(e) 注释框　　(f) 流向线

图 1-1　常用流程图符号

判断框:根据条件判断算法继续执行的走向,框内应填写条件。如图 1-1(c)所示。

处理框:表示执行计算赋值操作,框内用文字或符号表明具体实现的操作。如图 1-1(d)所示。

注释框:表示对算法中的某一操作作必要的备注说明,框内用文字说明备注信息。如图 1-1(e)所示。

流向线:表示算法的走向,箭头的方向表示算法执行的方向。如图 1-1(f)所示。

下面给出例 1-1,例 1-2,例 1-3 解题算法的流程图。

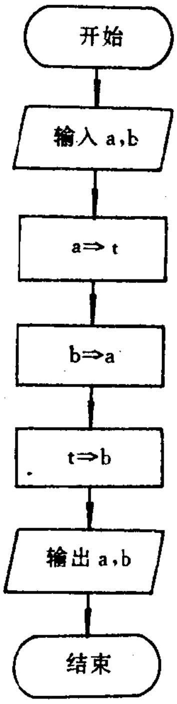

开始
输入 a,b
a⇒t
b⇒a
t⇒b
输出 a,b
结束

图 1-2　交换 a,b 两变量的值

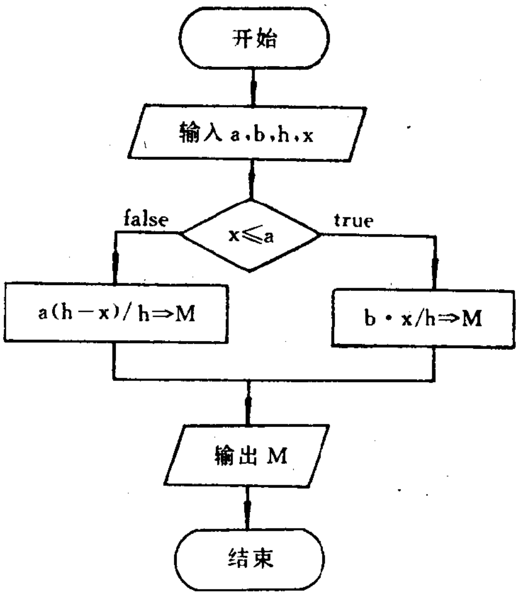

开始
输入 a,b,h,x
false
x≤a
true
a(h−x)/h⇒M
b·x/h⇒M
输出 M
结束

图 1-3　计算函数 M(x)的值

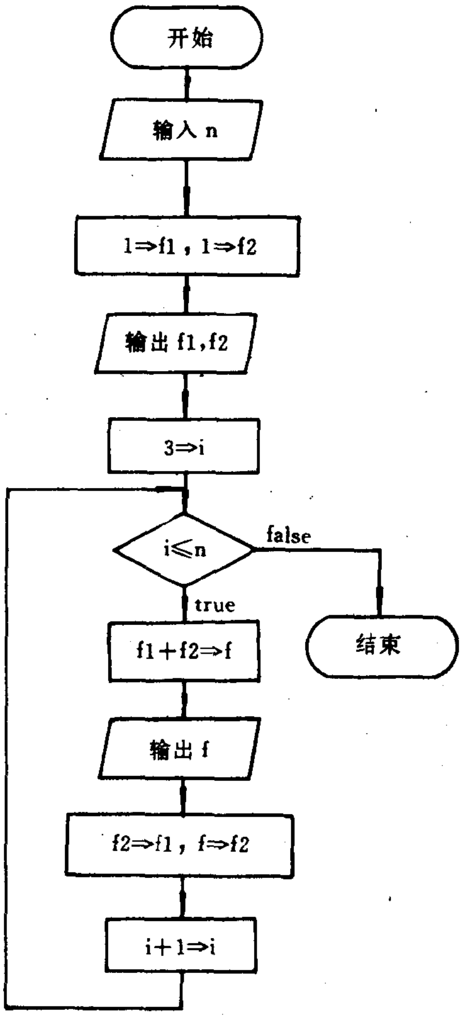

开始
输入 n
1⇒f1，1⇒f2
输出 f1,f2
3⇒i
i≤n
false
true
结束
f1+f2⇒f
输出 f
f2⇒f1，f⇒f2
i+1⇒i

图 1-4　计算 Fibonacci 数列前 n 项的值

以上介绍了算法的概念、特征及其流程图表示。实际上，设计算法还要涉及对算法所处理数据的研究。算法所处理的数据通常并不是杂乱无章的堆积，往往具有重要的关系，这就是数据结构的内容。实践证明，对数据进行合理的组织，选取合适的数据结构将会直接影响算法的选择及其执行效率。瑞士计算机科学家 N. wirth 用一个公式来揭示算法与数据结构之间的联系：算法＋数据结构＝程序。

数据结构是介于数学、计算机硬件、计算机软件之间的一门交叉学科，它研究的内容包括：数据的逻辑结构、物理结构以及数据结构上所施加的运算。

本书将程序设计中涉及到的有关数据结构的内容放在后续相关章节中讲述，届时，读者应给予足够的重视。

第二节　设计算法的基本方法和策略

为了获得一个有效的算法，读者必须了解一些解题的基本思想和方法。本节仅对构造算法所依据的一些基本策略（公式、方案、原则），即解题思想进行讨论，其目的是使读者遇到实际问

题时，能很快理出一条正规的解题思路，不致于束手无措，盲目乱凑。当然，这里讲述的内容还将在后续章节中结合具体问题作更加深入的研究。

一、迭代法

在科学计算领域，人们时常会遇到求解方程 $f(x)=0$ 或微分方程的数值解等计算问题。可是人们却很难或无法象用一元二次方程的求根公式那样的解析解法（又称直接求解法）。例如，一般的一元五次或更高次方程，几乎所有的超越方程，以及描述电磁波运动规律的麦克斯韦方程等，它们的解都无法用解析方法表达出来。为此，人们只能用数值方法（也称数值计算方法）求出问题的近似解，而解的误差人们可以估计和控制。

迭代法就是一种数值近似求解的方法，它既可以用来求解代数方程，又可以用来求解微分方程。其特点是使一个复杂问题的求解过程，转化为相对简单的迭代算式的重复执行过程。下面以求解方程 $f(x)=0$ 为例说明迭代法的基本思想。

首先把求解方程变换成为迭代算式 $x=g(x)$，然后由事先估计的一个根的初始近似值 x_0 出发，用迭代计算公式 $x_{k+1}=g(x_k)$ 求出另一个近似值 x_1，再由 x_1 确定 x_2，…，最终构造出一个序列 $x_0,x_1,x_2,\cdots,x_n,\cdots$，来逐次逼近方程 $f(x)=0$ 的根。

〔**例 1-4**〕 求方程 $x^3-x-1=0$ 在 $x=1.5$ 附近的一个根。

首先将方程改写成：$x=\sqrt[3]{x+1}$ （1）

用给定的初始近似值 $x_0=1.5$ 代入（1）式的右端，得到

$$x_1=\sqrt[3]{1.5+1}=1.35721$$

用 x_1 作为近似值代入（1）式的右端，又得到

$$x_2=\sqrt[3]{1.35721+1}=1.33086$$

重复同样步骤，可以逐次求得更精确的值。这一过程即为迭代过程，显然，迭代过程就是通过原值求出新值，用新值替代原值的过程。

对于一个收敛的迭代过程，从理论上讲，虽然经无限多次迭代可以得到准确解，但实际计算时，只能迭代有限次，因此，便有控制迭代次数的问题。

使用迭代法构造算法的基本方法是：首先确定一个合适的迭代公式，选取一个初始近似值以及解的误差，然后用循环处理实现迭代过程，终止循环过程的条件是前后两次得到的近似值之差的绝对值小于或等于预先给定的误差。并认为最后一次迭代得到的近似值为问题的解。

上述迭代方法称为近似迭代，除此之外还包括另一类，即精确迭代。所谓精确迭代是指算法本身提供了问题的精确解。一般说来，在构造精确迭代的同时已经可以确定所需迭代次数了，并且常常用它来控制迭代过程。例如，计算 $s=1+2+3+4+\cdots+1000$，可以确定迭代变量 s 的初始值为 0，迭代公式为 $s+i\Rightarrow s$，当 i 分别取 1，2，3，4，…，1000 时，重复计算迭代公式，迭代 1000 次后，即可求出 s 的精确值。

对于精确迭代问题，若结果有误差，通常不是算法本身造成的，而是因计算机用有限的字长表示实数造成的。

二、枚举法

枚举法亦称穷举法，它的基本思想是：首先依据题目的部分条件确定答案的大致范围，然

后在此范围内对所有可能的情况逐一验证，直到全部情况验证完。若某个情况使验证符合题目的全部条件，则为本题的一个答案；若全部情况验证完后均不符合题目的全部条件，则本题无答案。

利用枚举法解题大致需要以下步骤：

(1) 分析题目，确定答案的大致范围。

(2) 确定列举方法。常用的列举方法有：顺序列举，排列列举和组合列举。

(3) 作试验，直到遍历所有情况。

(4) 试验完后可能找到答案，证明题目有答案；也可能没找到答案，证明题目无答案。

利用枚举法解题的关键在于确定枚举范围和如何才能不重复，一个不漏，一个不增地逐一列举答案所在范围的所有情况。

枚举法常用来解决“是否存在”，“有多少种可能”，求解不定方程等类型的问题。利用枚举法设计算法大多以循环结构实现，其特点是算法结构简单，容易理解，但执行算法的时间开销较大。

三、递推法

所谓递推法，是指未知项与已知项之间存在某种关系，借助于已知项能一项一项地推出未知项。例如，求 n 的阶乘的算法存在如下递推过程：

$f(0)=0!=1$

$f(1)=1!=1\times0!=1$

$f(2)=2!=2\times1!=2$

$f(3)=3!=3\times2!=6$

………

$f(n)=n!=n\times(n-1)!=n\times f(n-1)$

要计算 15!，可以从递推初始条件 $f(0)=1$ 出发，应用递推公式 $f(n)=n\times f(n-1)$ 逐步求出 $f(1)$、$f(2)$、…、$f(14)$、最后求出 $f(15)$ 的值。

又如，Fibonacci 数列存在递推关系：

$F(1)=1$

$F(2)=1$

$F(n)=F(n-1)+F(n-2)$

若需要得到第 50 项的值，可以由初始条件 $F(1)=1$，$F(2)=1$ 出发，利用递推公式逐步求出 $F(3)$、$F(4)$，…，最后求出 $F(50)$ 的值。

读者也许会问若用通项公式来计算 $F(50)$ 的值不是更方便吗？事实上，有些问题要找出通项公式是相当困难的，并且即便找到计算也并非简便。例如，Fibonacci 数列的通项公式为：

$$F(n)=[((1+\sqrt{5})/2)^{n+1}-((1-\sqrt{5})/2)^{n+1}]/\sqrt{5}$$

显而易见，寻找这样的通项公式是相当不易的，并且利用上式计算 $F(n)$ 也相当费力。与此相反，若利用递推初始条件和递推公式进行计算就方便多了，尤其是计算机擅长重复处理。因此，当每步递推操作具有同一规律时，利用重复处理实现递推，则会使算法的执行十分有效。

四、递归法

递归也是构造算法的一种基本方法。如果一个过程直接或间接地调用它自身，则称该过程是递归的。例如，有许多递归定义的函数：

$$x^n = \begin{cases} 1 & n = 0 \\ x \cdot x^{n-1} & n > 0 \end{cases}$$

$$n! = \begin{cases} 1 & n = 0 \\ n(n-1)! & n > 0 \end{cases}$$

递归过程必须有一个递归终止条件，即存在“递归出口”。无条件的递归是毫无意义的。在阶乘的递归定义中，当 $n=0$ 时定义为 1，这就是阶乘递归定义的递归出口。

递归与递推是既有区别又有联系的两个概念。递推是从已知的初始条件出发，逐次递推出最后所求的值。而递归则是从需求的函数本身出发，逐次上溯调用其本身的求解过程，直到递归的出口，然后再从里向外倒推回来，得到最终的值。一般说来，一个递推算法总可以转换为一个递归算法。

递归的概念首先在数学领域出现，然而，按递归方法设计算法的策略不仅适用于计算数学问题，而且也适用于非数值运算领域。如递归可用于检索过程的求解。

实际上，递归算法在执行时间上往往比非递归算法要付出更多的代价，但尽管如此，因有很多问题的数学模型或算法设计方法本来就是递归的，用递归过程来描述它们不仅非常自然，而且证明算法的正确性也比相应的非递归形式容易很多，因此，递归仍不失为一种强有力的算法设计方法。

五、分治法

解一个复杂的问题时，尽可能地把这个问题分解为较小部分，找出各个的解，然后再把各部分的解组合成整个问题的解，这就是所谓的分治法。

使用分治法时，往往要按问题的输入规模来衡量问题的大小。若要求解一个输入规模为 n 且它的取值又相当大时，应选择适当的设计策略将 n 个输入分成 k 个不同的子集合，从而得到 k 个可分别求解的子问题，其中 $1<k\leqslant n$。在求出各个子问题的解之后，就可找到适当的方法把它们合并成整个问题的解，分治法便应用成功了。如果得到的子问题相对来说还太大，则可再次使用分治法将这些子问题分割得更小。

在很多考虑使用分治法求解的问题中，往往把输入分成与原问题类型相同的两个子问题，即 $k=2$。

分治法在设计检索、快速分类选择等问题的算法中是很有效的，并得到广泛应用。

六、回溯法

回溯法是设计算法中的一种基本策略。在那些涉及到寻找一组解的问题或者满足某些约束条件的最优解的问题中，有许多可以用回溯法来求解。下面以著名的八皇后问题说明回溯法的基本思想。

八皇后问题是指在一个 8×8 棋盘上放置八个皇后，且使每两个之间都不能相互“攻击”，即它们中的每两个都不处于同一行同一列及同一条斜线上，问共有多少种摆法。图 1-5 便是八

皇后问题的一个解(或称一个成功布局)。

我们把棋盘的行和列编上 1 至 8 的编号，这些皇后也给以 1 至 8 的编号。由于每一个皇后应在不同的行上，可以假定皇后 i 将放在行 i 上。因此，八皇后问题的全部解可以由形如 $(x_1,x_2,\cdots,x_8)$ 的 8 元组来表示，其中 x_i 表示第 i 行皇后所处的列数。显然，这个问题的一个约束条件为：$s_i=\{1,2,\cdots,8\}$，$1\leqslant i\leqslant 8$。此外，还有两个约束条件：没有任何两个皇后在同一列(即，没有任何两个 x_i 相等)，而且没有任何两个皇后在同一斜线上。这两个约束条件意味着所有的解只能是 $\{1,2,3,4,5,6,7,8\}$ 的某些排列。把图 1-5 所示的解表示为一个 8 元组就是 $(4,6,8,2,7,1,3,5)$。

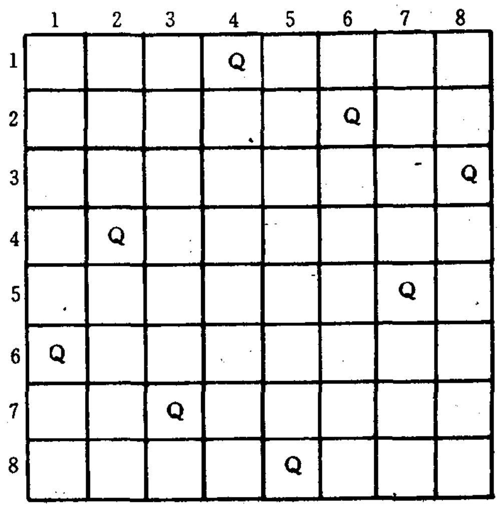
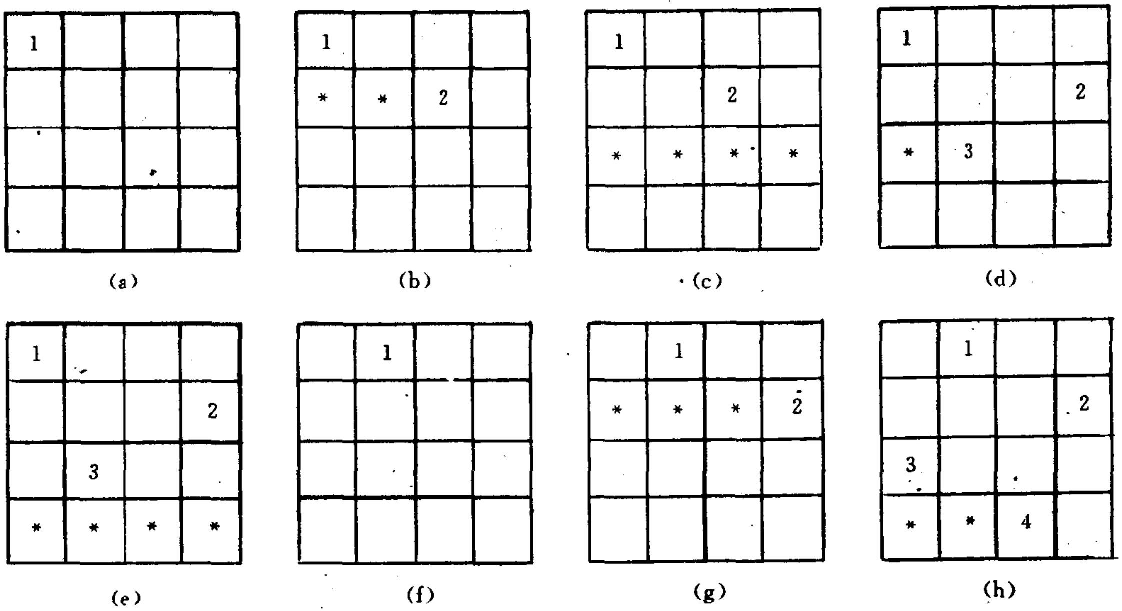


图 1-5 八皇后问题的一个解

如果一个问题的解能够表示成一个 n 元组 $(x_1,x_2,\cdots,x_n)$，其中 x_i 取值于 s_i，并且解及其子集(称为部分解)应满足一定的约束条件，则回溯法的基本思想是：若已有满足约束条件的部分解为 $(x_1,x_2,\cdots,x_i)$ $(i<n)$，则添加 $x_{i+1}\in s_{i+1}$，并检查 $(x_1,x_2,\cdots,x_i,x_{i+1})$ 是否满足约束条件，若满足再添加 $x_{i+2}\in s_{i+2}$；若对所有的 $x_{i+1}\in s_{i+1}$，都不能满足约束条件，则去掉 x_i，回溯到 $(x_1,x_2,\cdots,x_{i-1})$，添加那些尚未考察过的 $x_i\in s_i$，看其是否满足约束条件。如此反复进行，直到得到解或证明无解为止。

为使读者更好地理解上述思想，图 1-6 示意了用回溯法求四皇后问题的一个成功布局的例子。

图中的星号“*”表示曾试图放置一个皇后，但由于受到另一皇后的攻击而放弃了的位置。在(b)中，第 2 个皇后被放在第 1、第 2 列而最后放置在第 3 列上。在(c)中，算法把四列都察看



(a) (b) (c) (d) (e) (f) (g) (h)

图 1-6 四皇后问题的一个回溯解

了，但无法将下一个皇后放在一个位置上，此时进行回溯。在(d)中，第2个皇后被移到下一个可能的列，即第4列，然后将第3个皇后放在第2列上。图中的(e)——(h)则显示了用回溯法一直找到一个解所进行的基本步骤。

可以看出，回溯法是根据约束条件逐步扩展部分解，在扩展过程中可能需要回溯去修改部分解中前面的结点。

在实际问题的算法设计过程中，许多涉及分派或搜索问题的求解过程都要用到回溯搜索的控制策略。例如，给n个人分派n件工作，把工作j分配给第i个人的成本为cost(i,j)，在给每个人分派一件不同工作的情况下，求如何使总成本最小。求解该问题就用到回溯法。

以上介绍了几种构造算法的基本方法和策略，其目的是向读者提供一些处理不同类型问题的思考方法。对于数值计算问题，经过多人的研究，已经设计出比较成熟且被实践证明是有效的算法，程序设计时只需根据需要选择适用的算法即可。但选用时应当对算法所涉及的有关问题有所了解。

对于类似检索、统计、分类、插入、删除、修改等非数值计算问题，由于问题差别较大，要求各异，因此难以提供现成的、通用的算法。读者应当掌握设计算法的思维方法，触类旁通，举一反三，设计出自己所需的算法。

第三节　PAD的基本原理与图式

一、PAD的基本原理

当前，软件设计已进入软件工程时代，并产生了一些以图形为工具的软件设计方法。PAD(Problem Analysis Diagram，意为问题分析图)是其中最富生命力的一种软件详细设计表现法，它采用二维树形结构来描述程序的逻辑。

与流程图相比，PAD可以简洁地表现程序逻辑，即用PAD描述程序的流程能使程序结构一目了然。根据PAD编写程序对程序员的程序设计风格具有较强的约束性。此外，它易学、易读、易理解，便于编程、调试、审查和维护。

PAD使用的符号如图1-7所示。它们分别用于描述处理、选择、重复(循环)、语句标号、定义以及子程序。

PAD的核心思想是自顶向下设计和反复地逐步求精。“自顶向下设计”的宗旨是“分治”。即对复杂的问题并不马上着手解决“怎么做”，而且先解决“做什么”，也就是需要把复杂的问题分解成若干个子问题，然后对子问题再实施“分治”，直到所得到的子问题很明确，能够很方便地实现“怎么做”为止。“逐步求精”是指在上述设计过程中，有时会发现某一层次的“分治”不尽合理，这时就应该由底向上对上一层的“分治”方案进行修改，补充和加工。因此，可以把逐步求精的过程理解为是一种自底向上不断修改和补充的自顶向下的设计思想。

二、PAD的基本图式

如前所述，任何一个问题的求解算法都可以用顺序、选择、循环三种基本结构构造出来，将它们互相嵌套可以构造出结构更为复杂的算法。按三种基本结构构造的程序称为结构化程序。为支持程序的结构化设计，PAD提供了以下三种基本图式：

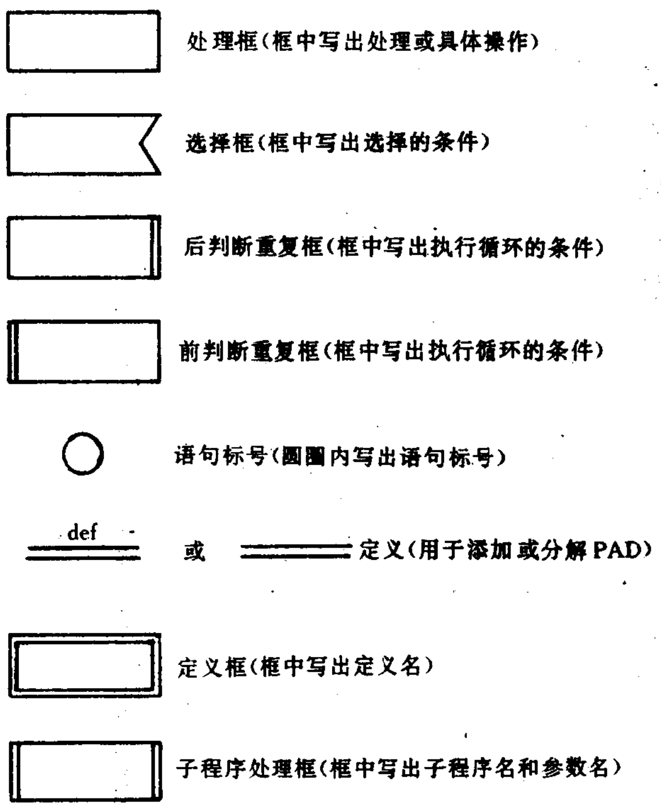

处理框（框中写出处理或具体操作）
选择框（框中写出选择的条件）
后判断重复框（框中写出执行循环的条件）
前判断重复框（框中写出执行循环的条件）
语句标号（圆圈内写出语句标号）
def
或
定义（用于添加或分解PAD）
定义框（框中写出定义名）
子程序处理框（框中写出子程序名和参数名）

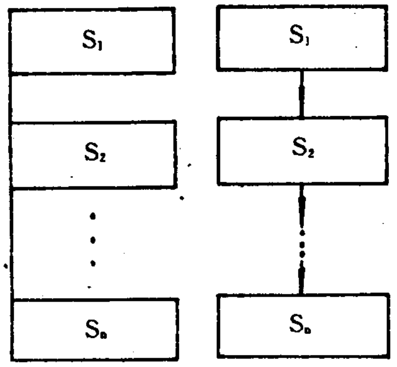
图 1-7　PAD 使用的符号

顺序型结构　处理两个或两个以上的操作序列。

选择型结构　处理两个操作时，根据条件成立与否，需要从中选择其一执行。

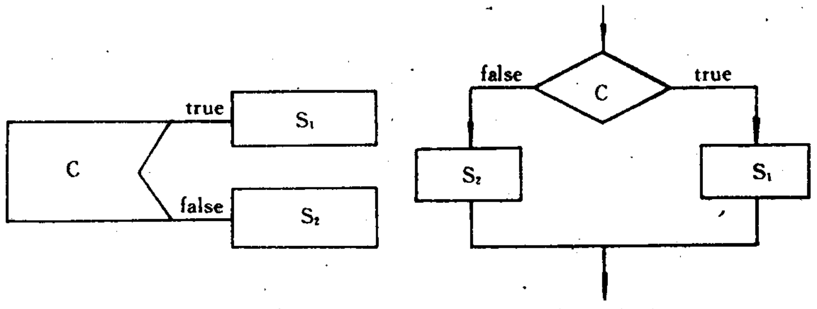
循环型结构　可细分为三种类型：① 当型循环结构，即条件成立时就重复执行。② 直到型循环结构，即重复执行，直到条件满足为止。③ 计数型循环结构，即根据循环控制变量（var）的初值（e1）、终值（e2）、增量（e3，亦称步长）事先可确定重复执行次数的循环。

上述基本结构的 PAD 图式描述与流程图的比较分别见图 1-8，图 1-9，图 1-10。图中 True（真）表示条件（c）成立，false（假）表示条件（c）不成立。图 1-10（a）、（b）、（c）分别表示当型、直到


S₁
S₂
Sₙ
S₁
S₂
Sₙ

PAD 图式描述　流程图描述

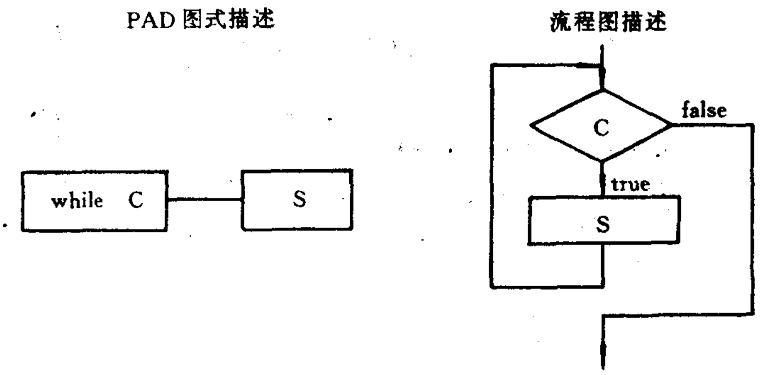
图 1-8　顺序型结构的比较


C
true
S₁
false
S₂
false
C
true
S₂
S₁

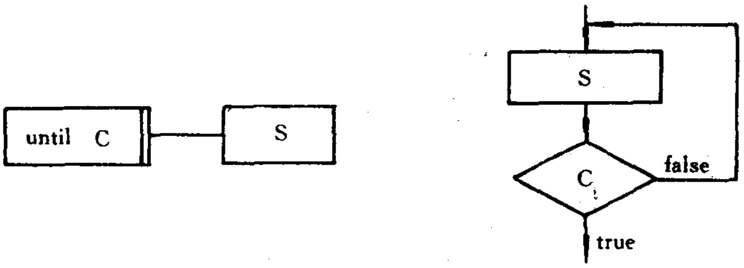
PAD 图式描述　流程图描述

图 1-9　选择型结构的比较

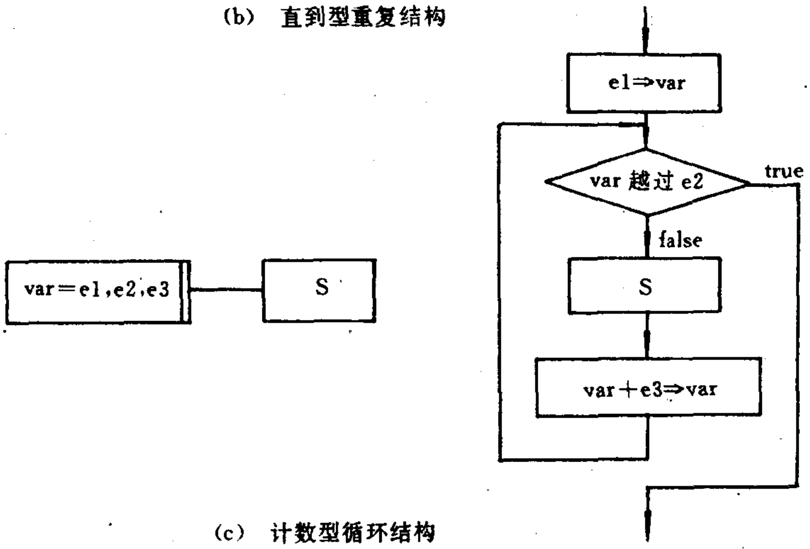
型和计数型循环结构。


PAD 图式描述
流程图描述
while C
S
C
false
true
S

(a) 当型重复结构


until C
S
S
C
false
true

(b) 直到型重复结构


var=e1,e2,e3
S
e1⇒var
var 越过 e2
true
false
S
var+e3⇒var

(c) 计数型循环结构

图 1-10 循环型结构的比较

三、PAD 的画法

PAD 图是一种二维树型结构的软件表示法。纵向表示顺序执行信息，横向描述条件分支或循环嵌套执行信息。图 1-11 和图 1-12 所示的 PAD 图分别表示第一节例 1-2 和例 1-3 的求解算法。

画 PAD 图时，读者可以从定义一个程序的顶层开始，然后再用 def 符号添加详细。符号

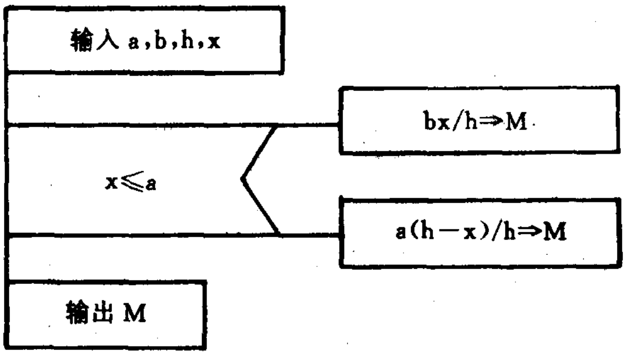

输入 a,b,h,x
bx/h⇒M
x≤a
a(h−x)/h⇒M
输出 M

图 1-11　计算函数 M(x)的 PAD 图

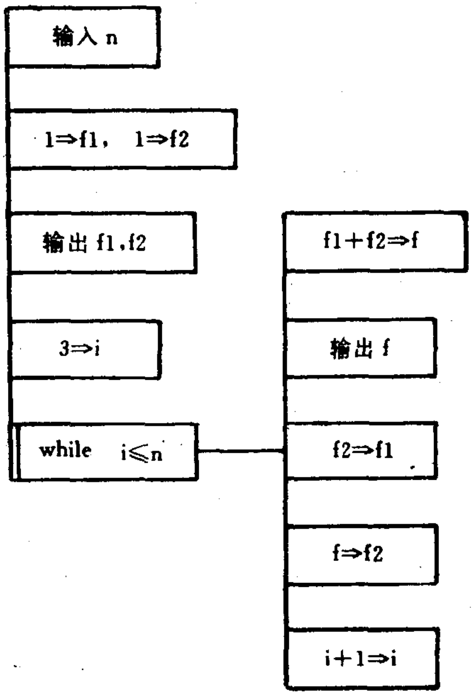

输入 n
1⇒f1,　1⇒f2
输出 f1,f2
3⇒i
while　i≤n
f1+f2⇒f
输出 f
f2⇒f1
f⇒f2
i+1⇒i

图 1-12　计算 Fibonacci 数列前 n 项的值

"def"用于定义 PAD 的详图，其画法如图 1-13 所示。

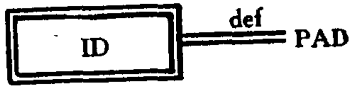


图 1-13　"def"符号的画法

其中 ID 为所定义的 PAD 的名称，如"交换 x，y 的值"。该符号的使用方法有：①用于表示自顶向下设计各个层次的详细步骤；② 用于把较大的 PAD 分解为较小的 PAD。③ 用于对某一处理添加 PAD，或者实现上下页的连接。

画 PAD 图时应注意以下事项：

(1) 当选择框后的某一分支处理为空时，可以省略该分支及其处理框。

(2) 规定画判断框时，条件成立(True)执行处于上端的处理框；反之执行处于下部的处理框。框外侧的"True""false"可省略不写。

(3) 横向连接 PAD 框时，选择框和循环框可以连接任一种 PAD 框。

(4) 纵向画 PAD 时，可画任何一种框。如处理框、选择框、循环框、定义框或子程序框等。

第四节　基本 PAD 的编程方法

一般来说，用计算机解决一个具体问题，大致需要以下几个步骤：首先要对具体的问题进行分析和抽象，提出操作的对象(如输入数据，或者计算所产生的中间结果、最后结果的表示方法等)，找出这些操作对象之间含有的关系，选择合适的数据结构，并构造出一个适当的数学模型，然后设计一个解此数学模型的算法，绘出 PAD 图，最后编出程序，进行审查、调试直到获得最终解答。

可见，将 PAD 图描述的算法转换为程序的编码过程是程序设计中的重要环节。下面以 True BASIC 语言为例，说明其转换原则。

将 PAD 图转换为 True BASIC 程序应遵循的原则是：把 PAD 图视为横向生长的树，并沿着树迹方向前进。当遇到处理框时，应根据其处理特征，选用相应的语句。例如，执行计算赋值时，选用 LET 语句；执行完成某一功能的子程序时，选用 CALL 语句。

遇到选择框时，由于 True BASIC 提供有多种格式的 IF 语句以及 SELECT CASE 语句，因此应根据具体情况灵活选用 IF 语句，或者选用 SELECT CASE 语句。

对于循环处理框，鉴于 True BASIC 提供有 DO 和 FOR 两种循环结构，而前者循环控制条件又分为当型(while 型)和直到型(until 型)两种，因此，遇到循环框时，根据具体情况选用相应的 DO 循环结构，或者选用 FOR 循环结构。

沿树迹结束时，编码过程即告完成。

〔例 1-5〕　验证哥德巴赫猜想

问题的由来：1742 年 6 月 7 日德国数学家哥德巴赫(C. Goldbaeh 1690——1764)，在给欧拉的信中写道：“任何大于 4 的偶数均可以表示为两个奇素数之和吗?”同年 6 月 30 日欧拉复信道：“任何大于 4 的偶数都是两个奇素数之和，这一猜想，我还不能证明，但我确信无疑地认为这是完全正确的定理。”这就是直到现在尚未被肯定或否定的数论中的世界著名难题——哥德巴赫猜想。

我国著名数学家陈景润等人认为这一猜想是肯定的可能性很大，并在这一领域作出突出贡献。这个问题目前最好的结果是：每一充分大的偶数都是一个素数及一个不超过二个素数的乘积之和。

下面编程验证这一猜想。首先介绍有关的数学知识。

定义：　一个大于 1 的正整数，只能被 1 和它本身整除，不能被其他正整数整除，这样的正整数叫做素数(亦称质数)。

例如，2，3，5，7，11，13，17，19 都是素数，而 3，5，7，11，13，17，19 是奇素数。

引理：　如果 a 是一个大于 1 的整数，而所有$\leqslant \sqrt{a}$的素数都除不尽 a，则 a 是素数。

引理：　有无限多个素数。

算法设计：如何判断 1 个大于 2 的自然数 x 是否为素数是验证哥德巴赫猜想首先要解决的问题。由引理可以推出：如果所有$\leqslant \sqrt{x}$的整数都除不尽 x(x>2)，则 x 是素数。由此，可写出算法：

设 flag_p 为 x 是否是素数的测试标志(若 x 是素数,则令 flag_p 的值为 1,否则为 0),i 是试除的除数。

```
s1:  1⇒flag_p;
s2:  2⇒i;
s3:  如果 i>int(sqr(x)),转向 s6;否则执行 s4;
s4:  如果 int(x/i)=x/i,作 0⇒flag_p,转向 s6;否则执行 s5;
s5:  i+1⇒i,转向 s3;
s6:  测试完毕,算法结束。
```

其中,“int(sqr(x))”表示计算 $\sqrt{x}$ 后取整,“int(x/i)=x/i”表示 x 能被 i 整除。上述算法的 PAD 图如图 1-14 所示。

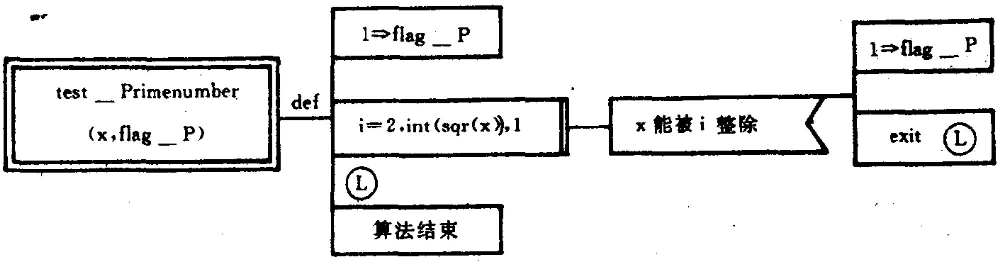


图 1-14 测试 x 是否为素数的 PAD 图

下面以 6～5000 中的偶数为例说明验证哥德巴赫猜想的算法设计。

设 n 为区间[6,5000]的一个偶数,根据题意,须寻找奇素数 m, m1,使算式 n=m+m1 成立。若 n 遍历区间中的所有偶数,均使算式成立,则说明哥德巴赫猜想是正确的;当 n 取区间中的某个偶数时,无法找到使算式成立的两个奇素数 m,m1,则表明哥德巴赫猜想是错误的。

为验证 n 可以表示为两个奇素数之和,首先取序列 3,5,7,9,11,…,n/2 中的第 1 个数作为 m 的值,再检测 m 是否为素数,若 m 是素数,则作 n−m⇒m1,接着再检测 m1 是否为素数。若 m1 是素数,表明算式成立。若 m1 不是素数,则应使 m 取序列中的下一个值,重复上述处理过程;假如 m 遍历序列中的所有数,均无法使算式成立,则表明哥德巴赫猜想是错误的,验证过程即可结束。

图 1-15 给出了验证哥德巴赫猜想算法的 PAD 图。

True BASIC 程序为:

```
! ---------------------------------------------
! Verifit the C. Goldbaeh's guess
! (even number : 6 to 5000)
! Main program
LET item_counter=0
FOR n=6 to 5000 step 2
      LET flag_success=0
      FOR m=3 to n/2 step 2
            CALL test_primenumber(m, flag_p)
```

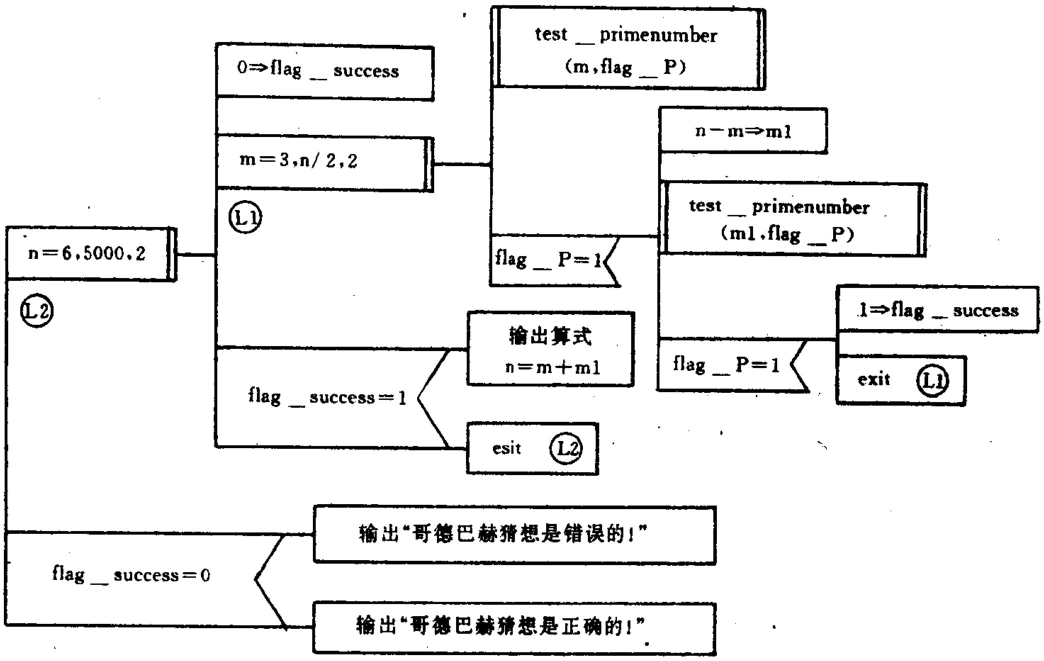

n=6,5000,2
L2
0⇒flag _ success
m=3,n/2,2
L1
test _ primenumber (m,flag _ P)
flag _ P=1
n−m⇒m1
test _ primenumber (m1,flag _ P)
flag _ P=1
1⇒flag _ success
exit L1
flag _ success=1
输出算式 n=m+m1
esit L2
flag _ success=0
输出“哥德巴赫猜想是错误的！”
输出“哥德巴赫猜想是正确的！”

图 1-15　验证哥德巴赫猜想的 PAD 图

```
   IF flag_p=1 then ！ M is a prime number.
      LET m1=n−m
      CALL test_primenumber(m1， flag_p)
      IF flag_p=1 then ！ M1 is a prime number.
         LET flag_success=1
         EXIT FOR
      END IF
   END IF
NEXT m
IF flag_success=1 then
   LET format$=“####  =  ##  plus  ####       ”
   PRINT using format$ :n， m， m1；
   LET item_counter=item_counter + 1
   IF item_counter=20 then
      PAUSE 10
      LET item_counter=0
      PRINT
      PRINT
      PRINT
   END IF
ELSE
```

```
        EXIT FOR
    END IF
NEXT n
PRINT
PRINT
IF flag_success = 0 then
  PRINT "The C. Goldbaeh's guess is erroneous !"
ELSE
  PRINT "The C. Goldbaeh's guess is right !"
END IF
END

!------------------------------------------
! External subroutine
! Test prime number,
! if x is prime number then flag_p reture 1
! else flag_p return 0
SUB test_primenumber(x, flag_p)
    LET flag_p=1
    FOR i=2 to int(sqr(x))
        IF int(x/i) = x/i then
          LET flag_p=0
          EXIT FOR
        END IF
    NEXT i
END SUB
!------------------------------------------
```

程序说明：

(1) 为提高程序的可读性，程序中多次使用注释语句(以!开头的语句行)对程序功能进行注释。

(2) 鉴于验证区间的偶数较多，为提高输出显示效果，程序中利用了分页(每页输出20个算式)输出技术，且每显示一页，暂停10秒钟，以便于操作者观察。

(3) 程序中定义了子程序 test_primenumber，以测试参数 x 是否为素数。

习　题

1—1. 什么是算法？它具有哪些特征？为什么说一个算法至少有一个输出？

1—2. 用自然语言和流程图描述下列各题的算法。

(1) 某类货物以90公斤为界限收取运费，计算公式为：

$$F = \begin{cases} aw & w \leqslant 90 \\ 90a + (w - 90)b & w > 90 \end{cases}$$

试计算运费，其中 w 为货物重量，a、b 为收费单价。

(2)四位数3025有一个特殊的性质，它的前两位数字与后两位数字的和是55，而55的平方等于该数，试找出所有具有这种性质的四位数。

1—3. 简单叙述迭代法、枚举法、递推法、递归法、分治法、回溯法等构造算法的基本思想，并分别说明使用前四种基本方法设计算法的关键是什么？

1—4. 为支持程序的结构化设计，PAD 提供了哪三种基本图式？画出这些图式并解释其执行流程。

1—5. 设计下列各题的算法，并用 PAD 图描述：

(1) 将1—100之间能被3整除的数输出显示，并统计其个数。

(2) 任输入一个大于1的正整数 n，求下式之和：

$$sum = 1 + 2 + 3 + 4 + 5 + \cdots + n$$

1—6. 下面是求一元二次方程 $ax^2+bx+c=0$ 解的算法，试用 PAD 图描述。

(1) 当 $a=0$ 时

①若 $b\neq0$ 时，$-c/b\Rightarrow x$

②若 $b=0$ 时，有两种情况：

若 $c=0$，输出“x 无定根”。

若 $c\neq0$，输入“x 无解”。

(2)当 $a\neq0$ 时，

①若 $b^2-4ac>0$，计算两个不等的实根 x_1，x_2，并输出显示。

②若 $b^2-4ac=0$，计算两个相等的实根 x_1，x_2，并输出显示。

③若 $b^2-4ac<0$，计算两个共轭复根 x_1，x_2，并输出显示。

1—7. 简单说明由 PAD 图描述的算法转换为程序应遵循的原则。

第二章　True BASIC 基本知识

第一节　True BASIC 简介

True BASIC，意为“真正的”BASIC，是由 BASIC 语言的创始人，美国的两位数学教授 G·Kemeny 和 E·Kurtz，根据美国国家 BASIC 标准，设计的一种典型的结构化程序设计语言。他们于1985年推出 True BASIC 的第一版本——1.0版，经过约二年的实践和改进，于1987年推出了功能更强的2.03版。

True BASIC 是一种既可采用解释方式，又可采用编译方式的“两栖型”语言。在调试程序时，采用解释方式，易于查错、纠错，以提高程序的调试效率。程序调试通过后，可采用编译方式，生成可执行的文件，使用时只用它的可执行文件，以加快程序的执行速度。对“两栖型”的 True BASIC 来说，既有编译型语言执行速度快的优势，又有解释型语言调试程序效率高的长处。

True BASIC 的另一个特点是它的结构化。在 True BASIC 编程中一般不使用行号，抛弃了其他语言中使用的 GOTO 语句，这就防止了程序流向的前后跳动。它提供的子程序、自定义函数、库文件、图画结构及链接技术，特别适合于大型程序的结构化设计。

为此，True BASIC 正受到越来越多的人的欢迎。

本章将首先介绍 True BASIC 的程序格式和字符集、常量、变量、标准函数、表达式等基本知识，然后讲述 True BASIC 上机操作初步。

第二节　True BASIC 的程序格式

True BASIC 程序是用 True BASIC 语言描述的算法，具有一定的程序格式。第一章我们已给出一个 True BASIC 程序，下面再通过一个简单的例子，说明书写 True BASIC 程序应遵循的格式。

〔例2-1〕 计算 $\sum_{n=1}^{100} n$

```
! calculate sum=1+2+3+……+100
LET sum=0
FOR n=1 to 100
    LET sum=sum+n
NEXT n
PRINT "sum=";sum                ! output the result
END
```

```
—————————————————————— ex02-1 ——————————————————————
Ok. run
sum= 5050
Ok.
```

从上面的程序可以看出：

(1) 一个 True BASIC 程序由若干个语句行所组成。

① 每个语句行由字母，数字，空格及其他允许的符号构成，一个语句行最多可有32767个字符。

② 每一个语句都要以一个关键字开始，如 LET，INPUT，PRINT 等。关键字用来区分不同的语句，其含义表示该语句的功能。

需要说明的是，若在解释执行方式下运行程序，每个语句前还可以有行号，但本书不介绍带行号的 True BASIC 程序设计方法。

(2) 一个完整的程序，其未尾应有一个 END 语句，执行到此语句，将停止该程序的运行。

(3) 程序的清晰性和可读性是衡量程序质量优劣的重要因素。为此，可以在程序适当的地方加入注释。注释语句通常以惊叹号开头，其后的字符串为注释内容。

第三节　True BASIC 字符集、常量、变量

一、字符集

True BASIC 字符集是 True BASIC 语言可使用的全部字符的总称。True BASIC 字符集包括：

英文字母：A～Z(大小写均可)

阿拉伯数字：0～9

特殊字符：+、-、*、/、∧、(、)、>、<、=、,、;、:、·、"、'、#、%、!、$、@、&、?、空格、下划线_。

以及机器所能识别的其他符号。

本书附录 A 列出了 IBM-PC 机可使用的全部字符。其中前128个被称为标准 ASCII 码字符，它们在任何种类的计算机上都是相同的。第129～256个字符则是 IBM-PC 机特有的，在程序中若出现了这些字符的显示和打印，其输出效果可能会因机器不同而异。

存储一个 ASCII 码字符，占用一个字节。

在汉字操作系统支持下，True BASIC 允许使用汉字，存储一个汉字，占用二个字节。

二、常　量

计算机处理的对象是数据，True BASIC 提供了两种基本的数据类型，数值型和字符型，每一类型又可分为常量和变量。

常量是指在程序内其值是固定不变的，可直接出现在表达式或置数语句中。

1. 数值常量(又称常数)

在 True BASIC 中一律采用十进制数。常数必须按规定的形式书写，一个常数中不能含有

空格，也不能用逗号作分节号，例如32768不能写成32␣768(␣意为空格，以后同)，也不能写成32,768。

常数的表示方法有以下两种：

(1) 日常记数法

每个数由一个正负号(正号可省略)和数字两部分组成，数字间可以带一个小数点，也可以没有。如3.1416，－4.8，362都是书写格式正确的数。采用日常记数法，一个整数的有效数字位最多可保留12位，一个带有小数部分的数，其有效数字位最多可保留6位。当一个整数超过12位，或者一个带小数部分的数在绝对值小于0.1或大于1000000时，将自动转为科学记数法来表示。

(2) 科学记数法

适合表示一个很大或很小的数，每个数由一个正负号(正号也可省略)，数字部分，指数部分共三部分组成。数字部分是不可缺少的，其中也可以带有一个小数点。指数部分的指数只能为整数。

如：　1100000000可写成11e8或11E8

－0.000003可写成－3e－6或－3E－6

56240000可写成5.624e＋7或5.624E＋7

不同型号的计算机所能表示常数的大小范围有所不同。对于 IBM-PC 机，常数的绝对值应在以下范围：$1.11254E-308 \leqslant |x| \leqslant 3.59539E+308$。如果数的绝对值大于3.59539E＋308，程序运行时将出现“overflow”(溢出)的错误信息，如果数的绝对值小于1.11254E－308时，机器则视该数为零。

在 True BASIC 中，存储一个数值型数据占用八个字节。

2. 字符常量(又称字符串)

字符串是由包含在两个双引号之间的若干个字符构成，例如“IBM-PC computer”，“CHINA”，“True BASIC”都是字符串。一个字符串中含有字符的总个数称为字符串的长度。在 True BASIC 中，字符串的最大长度为32766。在 CCDOS 支持下，程序中凡是可以出现字符串的地方，就可以出现汉字，一个字符串最多可含有16383个汉字。

双引号是用来确定字符串的起止界限的，因此，双引号本身不能算作字符串中的一个符号。如果要把双引号作为字符串中的一个符号，则必须在需要处连写两个双引号。如把一个英文句子 He said，“Good bye!”作为一个字符串时，应当写成“He said，”“Good bye!”“”。

空格也是一个常用字符，在计算字符串的长度时，双引号内的空格被视为一个有效字符。不含任何字符的字符串视为空串，用中间不含字符的两个双引号“”表示。

三、变量

程序执行时，其值可以改变的量称为变量。程序中的变量与内存中的存储单元相对应，存储单元中的数据即为变量的值。在 True BASIC 中，变量分为简单变量和下标变量，按照取值类型不同，它们又可分为数值型和字符型两类。数值型变量取值只能为数值，字符型变量取值只能为字符。关于下标变量的概念及用法将在第六章介绍，下面仅介绍简单变量。

为了便于识别和区分，每个变量都必须有一个唯一的名称，称为变量名。对存储单元中数据的运算处理，可通过对相应的变量名操作来完成，这也是高级语言区别于低级语言的一个显

著特点。True BASIC 中，简单变量的命名规则为：必须是以英文字母开头，由字母，数字，下划线组成的字符串，且长度不超过31个字符。例如 A，A_2，NUMBER 是合法的变量名，而2A，NUM␣3，x＃k 是非法的变量名。变量的命名最好能使人见文识义，如 AGE 表示年龄，SCORE 表示成绩等。

在 True BASIC 中，变量的类型定义采取不同的方式。对数值型变量，采用隐含的形式，如上面所列的合法变量名均为数值变量名，对字符型变量，则由变量名后跟一个 $ 符号来表示。如 NAME $，SEX $ 可分别表示姓名，性别等。

保留字因具有特殊含义和作用，一般是不允许作为变量名使用的。True BASIC 中的保留字如表2-1所示。

表2-1 True BASIC **保留字**

关键字	无参函数名		内部子程序名
ELSE	DATE	MAXNUM	DIVIDE
ELSE IF	DATE $	PI	PACKB
IF	DEF	RND	POKE
NOT	EXLINE $	TIME	
PRINT	EXTEXT $	TIME $	
REM	EXTYPE		

需要补充说明的是，除表中列出的以外，系统提供的其他标准函数名通常也视也保留字，因为在程序中若将其作为变量名，则在该程序内就无法使用系统定义的相应函数。

第四节　标准函数

函数是一种特定的运算。通过函数名，传递一个或多个参数，经过一定的运算或处理，获得一个输出。如果用户编写程序时，每遇到一个函数就要编出其特定的运算或处理程序，那将是十分繁琐的工作，也是一种沉重的负担。为此，True BASIC 提供了常用函数的内部程序供用户调用。使用时，用户只要写出其函数名并在紧接着的圆括号内给定相应的自变量，即可获得函数值。这种函数称为系统函数或标准函数。

True BASIC 提供的标准函数主要有数学函数，三角函数，字符串函数，日期和时间函数四类。本书只介绍前三类函数，关于日期和时间函数读者可参阅 True BASIC 使用手册。

一、常用数学函数

数学函数的自变量只能是常数或数值型的变量及表达式，其输出是一个数值。

表2-2 **常用数学函数**

函　数	含　义	调　用	输　出
ABS(x)	$\|x\|$，求 x 的绝对值	ABS(3.14) ABS(0) ABS(−9.5)	3.14 0 9.5

表2-2(续)

函　　数	含　　义	调　　用	输　　出
SQR(x)	$\sqrt{x}$,求 x 的平方根,$x \geq 0$	SQR(2) SQR(144)	1.41421 12
LOG(x) LOG2(x) LOG10(x)	$L_n x$,求以 e 为底的自然对数 $LOG_2 x$,求以2为底的对数 Lg(x),求以10为底的常用对数	LOG(1) LOG2(10) LOG10(0.01)	0 3.32193 −2
EXP(x)	e^x,求 e 的 x 次幂	EXP(1)	2.71828
SGN(x)	取 x 的符号 $X>0$　取值为1 $x=0$　取值为0 $x<0$　取值为−1	SGN(9.9) SGN(0) SGN(−4.2)	1 0 −1
INT(x)	求不大于 x 的最大整数	INT(3.9) INT(−2.1)	3 −3
ROUND(x)	求经过四舍五入后的整数,等价于 INT(x+0.5)	ROUND(3.15) ROUND(−5.6)	3 −6
ROUND(x,n)	$n \geq 0$时,x 值保留 n 位小数,且对第 n+1位进行四舍五入; $n<0$时,x 只取整数,且在小数点左边第$\|n\|$位进行四舍五入	ROUND(5.235,1) ROUND(18.56,0) ROUND(16.42,−1)	5.2 19 20
MAX(x,y)	求给定的两个数中的大数	MAX(−4,3)	3
MIN(x,y)	求给定的两个数中的小数	MIN(4,−7)	−7
MOD(x,y)	得到 x 除以 y 的余数,$y \neq 0$,结果在0与 y 之间	MOD(1,3) MOD(−1,3) MOD(3,−2)	1 2 −1
PI	π的常数值	PI	3.14159
RND	产生一个0～1区间的随机数。若在调用前,执行一条 RANDOMIZE 语句,可得到不同的随机数系列		
MAXNUM	得到机器所能表示的最大数		

二、三角函数

三角函数中的自变量 x 允许使用不同的单位(度或弧度)。若未加说明时(如表2-3所示),角的单位为弧度。

表2-3　　**三角函数**

函　数	含　　义	调　　用	输　　出
SIN(x)	求 x 的正弦	SIN(0) SIN(PI/2)	0 1
COS(x)	求 x 的余弦	COS(0) COS(PI)	1 −1

表2-3(续)

函　数	含　　义	调　　用	输　出
TAN(x)	求 x 的正切	TAN(PI/4) TAN(0)	1 0
ATN(x)	求 x 的反正切,函数值为弧度	ATN(0) ATN(PI)	0 1.26263
DEG(x)	把 x 的弧度值转换为角度值	DEG(PI/4)	45
RAD(x)	把 x 的角度值转换为弧度值	RAD(1)	0.017453
ANGLE(x,y)	求出 x 轴与原点(0,0)和坐标点(x,y)所成射线的夹角,x 和 y 不能同时为0	ANGLE(1,0) ANGLE(−1,0) ANGLE(0,1)	0 3.14159 1.5708

若要以度为单位时,可预先用 OPTION ANGLE DEGRESS 语句说明,其后角的单位为度。若要再以弧度表示时,则要使用 OPTION ANGLE RADIANS 语句说明,其后角的单位又被换成弧度。

三、字符串函数

字符串函数名最后不带 $ 时,函数值为一个数值,函数名最后带 $ 时,其值为一个字符串。

表2-4　**字符串函数**

函　数	含　义	调　用	输　出
LEN(x$)	测定 x$ 的长度,即 x$ 所含字符的个数	LEN(“BOOK”) LEN(“A”&“B”)	4 2
STR$(n)	将常数或数值表达式转换成相应的由数字构成的字符串	STR$(5−9) STR$(0.0000006)	−4 6.e−7
VAL(x$)	将数字字符串 x$ 转换成相应的数值,字符串若含首尾空格,则被忽略	VAL(“␣␣3.14␣␣” VAL(“1.994e+3”)	3.14 1994
ORD(x$)	将 ASCII 码字符集中的一个字符转换成相应的 ASCII 码值	ORD(“A”) ORD(“␣”)	65 32
CHR$(n)	将常数或数值表达式的值四舍五入后转换成一个相应的 ASCII 码字符,n 在0～255之间,n>255时,系统采用 MOD(n,256),把 n 折合到0～255之内	CHR$(66) CHR$(65.2) CHR$(65.6) CHR$(66+256)	B A B B
LCASE$(x$)	把 x$ 中的大写字母转换成相应的小写字母	LCASE$(“IBM-PC”)	ibm-PC
UCASE$(x$)	把 X$ 中的小写字母转换成相应的大写字母	UCASE$(“shanghai”)	SHANGHAI

表2-4(续)

函　　数	含　　义	调　　用	输　出
REPEAT $ (x $,n)	将 x $ 重复 n 次,构成一个新的字符串,n=0时,得到空串,n<0时,出错	REPEAT $ (“ * ”,5) REPEAT $ (“ab”,4)	* * * * * abababab
POS(x $,y $,n)	从 x $ 中左边第 n 个字符开始,查找第一次出现 y $ 时,y $ 的头一字符在 x $ 中的位置。n 为1时,可省略,如未找到 y $,函数值为0	POS(“SHANGHAI”,“HA”) POS(“SHANGHAI”,“HA”,3) POS(“SHANGHAI”,“AN”,5)	2 6 0
LTRIM $ (x $)	删除 x $ 中的左端空格	LTRIM $ (“␣␣ABC␣␣”)	ABC␣␣
RTRIM $ (x $)	删除 x $ 中的右端空格	RTRIM $ (“␣␣ABC␣␣”)	␣␣ABC
TRIM $ (x $)	删除 x $ 中左右两端的空格	TRIM $ (“␣␣ABC␣␣”)	ABC

True BASIC 程序中,凡是允许出现表达式的地方,均可以使用标准函数,但应注意函数值与表达式类型要一致,书写标准函数时,如有多个自变量,其间用逗号分隔。

第五节　表达式

True BASIC 允许进行算术运算,字符串运算,关系运算和逻辑运算,它们用来分别计算数值表达式,字符串表达式,关系表达式与逻辑表达式的值。本节仅介绍前两种表达式,后两种表达式将在第四章选择结构中介绍。

一、数值表达式

1. 算术运算符

True BASIC 提供的算术运算符有以下五种

符号	含义	示例
+	加(或正)	a+b
—	减(或负)	c—d
*	乘	m * n
/	除	x/y
∧	幂	5∧4

2. 数值表达式

用圆括号和算术运算符把常数,取值为数值的函数,数值型变量连接起来所得到的式子称为数值表达式(亦称为算术表达式),一个数或一个函数或一个数值型变量可视为最简单的数值表达式。

数值表达式的书写与一般代数式有所不同的,必须注意以下几点:

(1) 所有字符应写在同一水平线上。

(2) 乘号必须写成 *，且不能省略。

(3) 只允许使用圆括号( )，一层不够时，可使用多层，左右括号要成对使用。不允许使用方括号[ ]和花括号{ }。

(4) 常数 π 用 PI 代替。

下面是将代数式写成 True BASIC 数值表达式的几个例子。

代数式	True BASIC 数值表达式
$2\pi R^2$	2 * PI * R * R
$\frac{-b+\sqrt{b^2-4ac}}{2a}$	(-b+SQR(b * b-4 * a * c))/(2 * a)
$\frac{P(1+x)^n}{\sin 6x}$	P * (1+x) ∧ n/SIN(6 * x)
$a[x+3(y+Z)]$	a * (x+3 * (y+Z))

3. 数值表达式的运算规则

其运算顺序为：

()→函数→∧→ * ,/→+,-

即括号最优先，如有多层括号，内层比外层更优先，其次是函数运算，然后是算术运算。对于算术运算，乘方优先，乘、除次之，最后是加、减。同级运算，应从左到右依次进行。数值表达式的运算结果为一个数值。

二、字符串表达式

一个字符串的一部分，称为子字符串或子串。对字符量的运算有两种，一种是连接，一种是取子串。在 True BASIC 中，符号“&”为字符量的连接运算符。

用圆括号和“&”运算符把字符串，串变量，子串，取值为字符串的函数等连接起来所得到的式子称为字符串表达式，其运算结果为一个字符串。一个字符串或一个串变量或一个取值为字符串的函数都可视为最简单的字符串表达式。

例如：“IBM-PC”&“␣␣␣”&“COMPUTER”就是一个字符串表达式，其运算结果为一个新的字符串“IBM-PC␣␣␣COMPUTER”。

取子串的一般形式为：

〈字符串表达式〉 [a:b] 或〈字符串表达式〉 (a:b)

其中 a,b 为数值表达式，a 表示子串的起始位置，b 表示子串的终止位置。若 a<1，系统默认为1，若 b 大于整个串的长度，则用字符串长度来代替，如果 a>b，则得到一个空串。

还有一种取子串的特殊形式，即〈字符串表达式〉[a:MAXNUM]，表示取字符串表达式中第 a 个字符开始至最后一个字符为止的子串。

例如 x $ 为“THIS␣IS␣A␣BOOK”

那么 x $ [9:MAXNUM]运算结果为“A␣BOOK”。

可使用圆括号来改变字符串表达式的运算顺序。例如，“NAN”&“BEIJING”[4:7]是先取

子串后连接，运算结果为“NANJING”。若书写为(“NAN”&“BEIJING”)[4:7]则表示先连接后取子串，运算结果为“BEIJ”。

第六节　True BASIC 上机操作初步

一、True BASIC 的启动与退出

1. **启动** True BASIC

当机器处于 DOS 状态时，键入 HELLO 命令，便可启动 True BASIC。例如，系统当前驱动器为 A，将 True BASIC 2.03版系统软盘插入 A 驱动器中，关好驱动器门，并在 A>提示符之后键入 HELLO 并按回车键，即：

```
A>HELLO〈CR〉
```

稍等片刻，屏幕底部便显示出下列信息：

```
———————————————————————————————— Untitled ————————————————————————
True BASIC here.  Version 2.03
Copyright (c) 1988 by True BASIC, Inc.  All rights reserved.
Type SPLIT to see a menu of the function keys.
Ok.
```

“OK”是 True BASIC 的提示符，表示系统已进入 True BASIC 状态，用户可以输入 True BASIC 程序或命令。

2. **退出** True BASIC

使机器自 True BASIC 状态返回到 DOS 状态，称为退出 True BASIC。其操作是在“OK”提示符之后键入 BYE 命令并按回车键，如果 A 为当前驱动器，则屏幕上将出现 A>提示符。

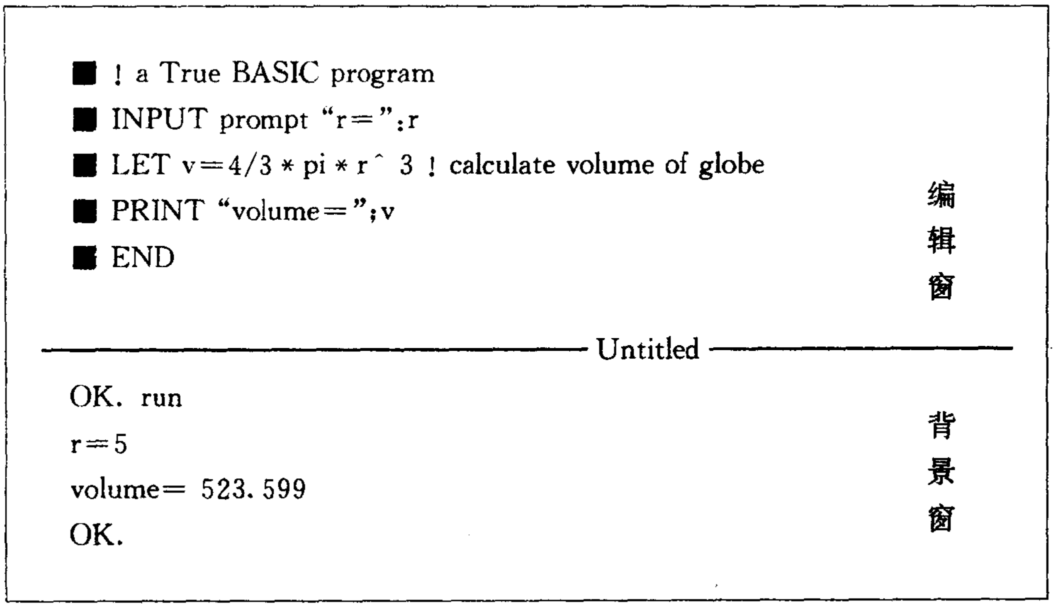
二、True BASIC 屏幕窗口

在 True BASIC 状态下，将屏幕分成上下两个区域，中间用一条横线分开，上部称为编辑窗口(editing Window)，下部称为背景窗口(history Window)。编辑窗供用户输入，编辑或显示 True BASIC 程序。背景窗也称命令窗，供用户键入 True BASIC 命令，显示程序的运行结果或出错信息，并可记录最近执行的命令或程序执行结果的痕迹。启动后编辑窗占17行，背景窗占7行，但用户可用 SPLIT 命令改变窗口的大小。两窗口的分界线中间，设置了一个文件名监视口，当用户编辑未命名的文件时，其中显示 Untitled，当用户存入或调出一个已知文件名的程序时，上述显示变成当前存入或调出的实际文件名。

光标是一闪烁的小亮线或块，宽度为一个字符，它指出下次键入的字符将在屏幕上的位置。光标处于哪个窗口，该窗口就成为当前工作窗，可以使用功能键 F1，F2来改变当前工作窗，按 F1键，光标移到编辑窗，按 F2键，光标移到背景窗。

使用彩色显示器时，可按 ctrl+F，ctrl+B，ctrl+E 键，分别选择光标所在窗口的前景，背景和边缘颜色。

窗口的状况如图2-1所示。



```
■ ! a True BASIC program
■ INPUT prompt "r=":r
■ LET v=4/3 * pi * r ^ 3 ! calculate volume of globe
■ PRINT "volume=";v
■ END
```

```
OK. run
r=5
volume= 523.599
OK.
```

图2-1 True BASIC 屏幕窗口

三、源程序的编辑

源程序的编辑工作是在编辑窗进行的。所谓编辑是指输入源程序，或对其进行修改。若输入一个新程序，应该首先在背景窗输入 NEW 命令，以清除内存中原有的程序，然后在编辑窗逐字逐行输入新程序的内容。在输入过程中，随时可对错误的输入进行修改。若要对以文件形式存储在磁盘上的某个程序进行编辑，则必须先用 OLD 命令将其调入内存，并显示在编辑窗内，再进行编辑。

True BASIC 的编辑是在全屏幕状态下进行的。此时，编辑窗相当于一张纸，光标相当于笔，用户可以使用移动光标键将光标移到任意指定位置处进行编辑操作。这些操作大多是由称之为键命令的特定键来完成的。表2-5和表2-6列出了在 IBM-PC 机上进行全屏幕编辑时移动光标键和常用编辑键及其功能，具体使用方法将在第十二章上机操作指导中详细介绍。

表2-5 **移动光标键及其功能**

键	功能	键	功能
→	右移一个字符位置	PgUp	向上翻页
←	左移一个字符位置	PgDn	向下翻页
↑	上移一行	Tab	右移一个字
↓	下移一行	Shift+Tab	左移一个字
Home	移至程序首行	Ctrl+←	移至行首
End	移至程序末尾	Ctrl+→	移至行尾

表2-6　　常用编辑键及其功能

键	功　能	键	功　能
Back Space	删除光标左边一个字符	ESC	删除光标左边的行内容
Del	删除光标处一个字符	Ctrl + End	删除光标右边的行内容
Ins	改变插入或替换状态	Del	光标处于行标志处时，删除当前行
Ctrl + Home	删除光标左边一个字	F7	取消当前行的修改，恢复当前行的内容
Ctrl + PgUp	删除光标处一个字	Enter	在光标处插入一个空行

四、解释，编译，汇集

True BASIC 允许采用解释和编译两种工作方式。调试程序时用解释方式，以提高调试效率。具体操作方法是在 OK 提示符之后执行 RUN 命令（或按 F9键），系统即进入解释方式，对程序逐句解释并执行，如果发现哪个语句有语法错误，则光标置于该语句处，并在背景窗显示相应的错误信息，待用户修改。反复执行这一过程，可快速地找出和改正程序中的语法错误。当程序不存在任何语法错误时，方可执行并得到运行结果。

为了提高程序的执行速度，可在程序调试通过之后，利用系统提供的编译方式，将其进行编译，以生成扩展名为 TRC 的编译代码文件。对 True BASIC 程序的编译是通过执行 COMPILE 命令实现的。编译完成后，执行 RUN 命令，即可运行编译代码文件并得到运行结果。此时，系统不需要对程序逐行进行解释，运行效率较高。

True BASIC 2.03版可将经编译得到的扩展名为 TRC 的文件汇集生成扩展名为 EXE 的文件。其具体操作参见第十二章。这种扩展名为 EXE 的文件可直接在 DOS 状态下执行，而不受 True BASIC 环境限制。

五、程序文件的保存与调用

当断电时，内存 RAM 中的信息就会消失。为了把内存中的程序保存起来，以便在需要时再次使用，True BASIC 允许用户使用 SAVE 命令，把内存中当前的程序以文件的形式保存在磁盘上。SAVE 命令的格式为：

```
SAVE  [〈盘符〉:]  文件名[·〈扩展名〉]
```

其中盘符的使用，文件名和扩展名的命名，要符合磁盘文件的有关规定，当省略扩展名时，系统会自动加上扩展名 TRU。

需要指出的是，在同一磁盘的同一目录下，不能存储两个或两个以上的同名文件。

若要使用存储在磁盘上的程序文件时，可用 OLD 命令将其调入内存，该命令格式为：

```
OLD  [〈盘符〉:]文件名[·〈扩展名〉]
```

执行这个命令，将把指定的程序调入内存，并在编辑窗显示其内容。

若用 OLD 命令把磁盘上的一个程序文件调入内存并修改后，还想重新存入原磁盘，应使用 REPLACE 命令，其格式为：

```
REPLACE  [[〈盘符〉:]  文件名[·〈扩展名〉]
```

此命令还可以不带文件名，这时只需键入 REPLACE 并回车，内存中的程序，则以同名写到磁盘的原位置上，同名是指由 OLD 命令提供的文件名。

习题

2—1. 称 True BASIC 为“两栖型”语言，具体指什么？它有何优点？

2—2. 简述 True BASIC 变量命名的规则，并且指出哪些是合法的变量名，哪些是不合法的变量名？如不合法，说明原因。

A,x1,3Y,BOOK1,SEX $,ABC

PRINT,NAME $,T # N,π,TIME $

2—3. 写出下列数学函数的值。

(1)ABS(−13.6)

(2)ABS(5.17)

(3)INT(−3.14)

(4)INT(7.96)

(5)SGN(−125)

(6)SGN(6.28)

(7)SQR(169)

(8)ROUND(1.2 * 4)

(9)MAX(3.5,−1)

(10)MOD(14,3)

2—4. 写出下列字符串函数的值。

(1)LEN(“BEIJING”)

(2)LEN(“BOOK”&“PEN”)

(3)ORD(“D”)

(4)CHR $ (74.2)

(5)LCASE $ (“FORTRAN77”)

(6)UCASE $ (“computer”)

(7)REPEAT $ (“ # ”,6)

(8)POS(“IBM-PC”,“PC”,3)

2—5. 将下列代数式转换成相应的 True BASIC 表达式。

(1)$\frac{2(y+2)}{3(x-4)}$

(2)B^2-4AC

(3)$2^m\sin^2 45°$

(4)$L_n(x+\sqrt{1+y^2})$

(5)$\frac{4}{3}\pi R^3$

(6)$\frac{\sqrt{D-\left(\frac{n}{2}\right)^2}}{e^3\cdot r^2}$

2—6. 将下列 True BASIC 表达式转换成相应的代数式。

(1)SQR(x * x−1)+1/(x+1)/(x+2)

(2)SQR(x * x−1)/(x+1) * (x+2) ∧ 3

(3)PI * R * R

(4) y−Exp(x) * SIN(PI/4)

(5)(M1 * M2)/LOG(5)+ABS(x)

(6)(y+2) ∧ 3/(x+4) ∧ (1/3)

2—7. 根据下列各题的题意，分别写出相应的 True BASIC 表达式。

(1) 以2为底数，12为真数的对数值。

(2) 30°的正弦与45°的余弦之和。

(3) R 为底半径,H 为高的圆锥体的体积。

(4) 利用 RND 函数,产生一个两位随机整数。

2—8. 简述 True BASIC 启动与退出的方法。

2—9. 简述编辑窗与背景窗的作用,如何改变窗口的大小?

2—10. 简述 True BASIC 程序文件的存取方法。

第三章　数据的输入与输出

前面介绍了 True BASIC 的一些基本知识，其中指出每个源程序都是由若干个语句行构成的。从这一章开始，我们将逐一介绍 True BASIC 的各种语句。熟悉它们各自的格式和功能，掌握其使用方法，是进行程序设计的基础。

计算机执行程序是对用户提供的原始数据进行自动化加工，然后向用户输出加工结果信息的过程。因此，一个源程序一般包括数据输入，运算处理，输出结果三部分。数据的输入输出要通过具体的输入输出语句来实现。它们是程序中最基本，最常用的语句。

本章将重点介绍 True BASIC 中的 PRINT，LET，INPUT 和 READ 语句，前者为屏幕输出语句，后三者是为程序中的变量提供数据的输入语句。最后讨论 True BASIC 程序设计的一般步骤及有关问题。

第一节　PRINT（屏幕输出）语句

第一章讲述的算法特征告诉我们，运行一个源程序总是应该有一个或多个输出的。PRINT 语句的作用就是将程序的运行结果或需要输出的中间结果显示在屏幕上。本节仅介绍屏幕输出，至于在打印机上如何输出，将在第九章和第十二章介绍。

PRINT 语句的一般格式为：

```
PRINT 〈表达式组〉
```

其中表达式组为若干个表达式，每个表达式称为一个输出项。该语句的执行过程是先计算出各表达式的值，然后将所得值传送到屏幕上显示。具体输出效果与输出项的类型及输出项之间的分隔符有关。

一、输出内容

1. 输出数值型数据

当输出项为数值表达式时，输出的是数值。输出时，每个数前有一个符号位，接着为数字部分。如果是正数和零，符号位为空，如果是负数，符号位为“－”。

例如，执行 PRINT 125＋314语句，显示结果为␣439，执行 PRINT －3e＋15＊1.2e－5语句，显示结果为－3.6e＋10。

几点说明：

（1）对于一个不超过12位的整数，系统将其作为一个整数处理，按日常记数法输出，若大于12位，则按科学记数法输出。

（2）对于带有小数部分的数，总位数不超过6位时，按日常记数法输出，且省略小数部分末尾的零；总位数超过6位时，最多只输出6位有效数字（对第7位四舍五入），其中经四舍五入后绝对值小于等于0.1或大于等于1000000的数通常以科学记数法输出。

(3) 科学记数法的输出形式为±d·dddddе±dd，每个 d 的位置上为一个十进制数字，但小数点左边的 d 位置上只能为一个非零数字，小数部分末尾的零也将省略。

(4) 对于带小数部分的数，如果需要输出的精度超过6位，可使用后续章节介绍的 PRINT USING 语句。

2. 输出字符型数据

当输出项为字符串表达式时，输出字符串，字符串数据项没有符号位。

例如，执行语句

```
PRINT "SHANGHAI"
```

输出结果为：

```
SHANGHAI
```

若执行语句

```
PRINT "True␣BASIC␣" & "PROGRAM"
```

输出结果为：

```
True␣BASIC␣PROGRAM
```

二、输出格式

PRINT 的输出格式有四种，即标准格式，紧凑格式，带 TAB(x)函数的指定格式和带 USING 的自选输出格式。下面仅介绍前两种输出格式的使用方法，后两种将在后续章节介绍。

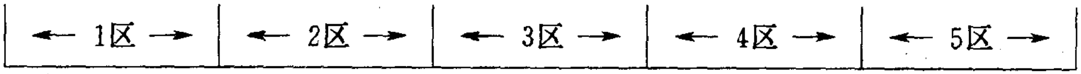
1. 标准格式

当 PRINT 后面的各输出项之间以逗号分隔时，则各输出项的值按标准格式（或称分区格式）输出。所谓标准格式输出，是指把屏幕上的一行（80位）等分成五个区，每个区为16位，各输出项的值依次从相应区的开始处输出。即第1项的值从一区起始位置（即从第1号位）开始输出，第2项的值从二区起始位置（即从第17号位）开始输出，其余依此类推。当输出项超过五项时，则自动换行，超过部分继续按区依次输出。

例如执行如下语句：

```
PRINT 5,10,15,-20,25,-30,35
```

输出结果为：



```
␣5              ␣10             ␣15             -20             ␣25
-30             ␣35
```

各区的首位输出数的符号，如为正数或零，该位为空格，如为负数，该位为"－"。一行最多输出五项数据，如超过五项数据，超过的项自动输到下一行。

当输出项为字符型，且所含字符个数超过16时，将按该项的长度分配两个或更多的区。其后一输出项将从已实际占有区的下一个区开始输出。例如，执行语句：

```
PRINT "ABCDEFGHIJKLMNOPQRST","UVW"
```

则输出结果为：

```
|  ←  1区  →  |  ←  2区  →  |  ←  3区…  |
ABCDEFGHIJKLMNOPQRST          UVW
```

2. 紧凑格式

当 PRINT 后各输出项之间以分号分隔时，则各输出项的值按紧凑格式输出。若输出项为数值型，则在输出数的符号和数字部分之后，自动留一个空位，紧接着再输出下一项；若输出项为字符型，输出字符串之后不留空位，紧接着输出下一项。

例如，执行以下语句：

```
PRINT 5;8;-4;-2;"ABC";"DEF"
```

输出结果为：

```
␣5␣␣8␣-4␣-2␣ABCDEF
```

以紧凑格式输出多个数据项时，只有当输出位数超过80位时，才自动换行。

在一个 PRINT 语句中，各输出项之间允许逗号和分号混合使用（即所谓混合输出格式）。这时逗号和分号各自所隐含的输出格式不变。

例如，执行语句：

```
PRINT "3*5=";3*5,"3+5=";3+5,"ABC"
```

其输出结果为：

```
|  ←  1区  →  |  ←  2区  →  |  ←  3区  →  |  ←…
3*5=␣15        3+5=␣8         ABC
```

3. PRINT 语句最后处有无标点的用法

（1）当 PRINT 语句最后处无标点符号时，本行各输出项的值输出后，系统都将自动换行，而与本行位置是否占满无关。后续的 PRINT 语句的输出项将从一个新行开始输出。

（2）当 PRINT 语句最后处为逗号时，本行各输出项的值输出后，系统不换行（若本行已占满除外），后续的 PRINT 语句各输出项的值将接着从本行尚未占用的头一个区开始输出。

（3）当 PRINT 语句最后处为分号时，本行各输出项的值输出后，如果本行未占满，系统不换行，后续的 PRINT 语句中各输出项的值，紧接在本行最末输出项的值之后输出。

例如，执行以下程序段：

```
PRINT 5+1
PRINT 125,
PRINT 4*(-1.5)
PRINT "3+5=";
PRINT 3+5
```

输出结果为：

```
 6
 125          -6
3+5=  8
```

4. 关键字 PRINT 之后无任何内容

执行此语句，只是将光标移到下一行，用以增加一次换行或输出一个空行。

〔**例3-1**〕

```
PRINT "x","y","z",
PRINT
PRINT 3,4,5
PRINT
PRINT "a","b","c"
PRINT -6,-7,-8
END
```

——————————————————— ex03—1 ———————————————

```
OK. run
x                 y                  z
  3                 4                  5


a                 b                  c
-6                -7                 -8
OK.
```

需要补充说明的是，PRINT 语句具有计算和屏幕输出功能。执行此语句，是把表达式的值计算出来并直接输出，该值并未保存，以后要用到时还须重新计算。

此外，在 True BASIC 中，PRINT 语句还可作为一个命令来使用。在背景窗 OK 提示符之后键入 PRINT〈表达式组〉并按回车键，则显示出各表达式的值。显然这在调试程序时查看某个变量或函数的值是很方便的。

第二节　LET(赋值)语句

程序中计算表达式的值，并把计算结果保存起来，以便多次利用，主要是通过对变量赋值来完成的。True BASIC 中的 LET 语句则具有计算和赋值的功能，它有如下两种格式：

LET〈变量〉=〈表达式〉

LET〈变量1〉,〈变量2〉,……=〈表达式〉

语句的作用是先计算出表达式的值，然后传送给一个或多个变量。

格式中的“=”称为赋值号，前一种格式是把表达式的值赋给一个变量，后一种格式是把表达式的值同时赋给多个变量，即给多个变量赋同一个值。

赋值号的左右两边必须保持类型一致，即赋值号右边为数值表达式时，则左边的变量必须是数值变量；赋值号右边为字符串表达式时，则左边的变量一定要为字符串变量。例如：

```
LET x=(-b+SQR(b*b-4*a*c))/(2*a)
LET i, j, k=3
LET capital $ ="Beijing"
LET book_name $ ="True BASIC" & "Programming"
```

另外，要特别指出的，LET 语句不是代数等式。赋值号与等号形同而义不同。等号静态地表示其两边的代数式具有“相等”的关系，而赋值号表示一种带方向性的操作。例如，LET I=I+1是合法的赋值语句，表示将名字为 I 的变量所对应的存储单元中的数值与1相加，将结果送回该单元中，使该单元中的数值累加了1。但在代数中，“I=I+1”显然是不存在的。况且，数学中的等号两边都可以是数，变量或代数式，两边对调后与原来的关系等价。而赋值号的左边必须是变量，右边可以是数，变量或表达式，两边不可以对调，即使两边都是变量，对调后所产生的结果与对调前一般是不相同的。

关于 LET 语句，还需要说明以下几点：

(1) 把字符串赋给串变量时，该字符串必须用双引号括起来。

(2) 赋值号左侧只能为一个或多个变量，多个变量间用逗号分隔。并且一个赋值语句只能使用一个赋值号。

如 LET A=3+2:B=4+5

LET x=y=z=5

都是错误的。

(3) 在一个程序中对同一个变量多次赋值，则新值冲去旧值，变量最终取最近一次赋的值。若赋值号两侧出现相同的变量，那么右侧的变量表示本次赋值前的值，而左侧的变量表示本次赋值后的新值。

(4) 可通过对子串的赋值操作，实现对字符串的部分修改，即对字符串的部分替换，插入和删除。

① 替换。在赋值号左侧放被替换的子串，右侧放一个用来替换的新字符串，两者的长度可以不等。

② 插入。对某字符串起始位置是1，结束位置是0的子串赋值，可以把一个字符串加到某字符串的开头；对起始位置是 MAXNUM，结束位置是0的子串赋值，可以把一个字符串加到某字符串的末尾；对一个起始位置大于结束位置的子串赋值，可以把一个字符串插入到某一个字符串之中。

③ 对某字符串中的一个子串赋以空串，可达到从某字符串中删除该子串的目的。

〔例3-2〕

```
! replace
LET A $ ="ABCD"
LET A $ [1:1]="M"
PRINT A $
LET B $ ="True BASIC program"
LET B $ [1:10]="FORTRAN"
PRINT B $
LET C $ ="IBM-PC"
LET C $ [5:MAXNUM]="computer"
PRINT C $
```

```
 ! insert
LET D$ =“BOOK”
LET D$[1:0]=“A  ”
PRINT D$
LET E$ =“IBM-”
LET E$[MAXNUM:0]=“PC”
PRINT E$
LET F$ =“SHANGHAI TV”
LET F$[9:6]=“  COLOR”
PRINT F$

 !  delete
LET G$ =“True BASIC”
LET G$[1:5]=“”
PRINT G$
END
```

——————————————ex03-2——————————————

```
OK. run
MBCD
FORTRAN program
IBM-computer
A BOOK
IBM-PC
SHANGHAI COLOR TV
BASIC
OK.
```

第三节　INPUT(键盘输入)语句

INPUT 语句称为键盘输入语句，也是一种为变量提供数据的语句，其作用是把从键盘上输入的数据传送到变量所对应的存储单元中去。

前面所述的赋值语句只能给变量赋一个确定的值，这个值要写在程序中，若要改变该变量的取值，则必须修改程序。然而，在实际应用中常会碰到这样的情况，需多次运行的同一程序或程序段中，变量的个数、类型、以及算法均不改变，只是改变变量的取值，若仍使用赋值语句为变量提供数据，变量的值每改变一次都得修改一次程序，这是很不方便的。若使用 INPUT 语句，为变量提供的数据不写在程序中，只是在运行程序时，临时从键盘上输入，当变量取值改变时，只要输入不同的数据即可，而程序不需要作任何改动。这对于编制那些参数变化的通用程序，其优点是明显的。

键盘输入语句的一般格式为：

```
INPUT  [PROMPT 〈提示字符串〉:]〈变量1〉[,〈变量2〉]…
```

其中[PROMPT〈提示字符串〉:]为可选项。若选择了该项，语句执行时，首先在屏幕上照原样显示出提示字符串，作为键入数据的提示信息，然后暂停程序的执行，以等待用户自键盘为变量提供数据。若未选用可选项，语句执行时，屏幕上则显示出一个问号“?”，然后暂停，以等待用户的输入。

其中〈变量1〉[,〈变量2〉]…是需要提供数据的变量名，可以是一个或者多个，可以是数值变量，也可以是字符串变量。自键盘输入的数据与语句中的变量在个数、类型上要一一对应。

例如：

INPUT a 屏幕显示?后，输入一个数值，然后按回车键。

INPUT PROMPT “name, sex, age=”:name $,sex $,age

屏幕显示提示字符串后，要求输入三个数据，前二个为字符串，最后一个为数值，彼此间用逗号分隔，然后再按回车键。

〔**例3-3**〕 根据提供的半径 r，求圆的周长 d 和面积 s。

其程序为：

```
! calculate circumference and area of circle
INPUT r
LET d=2 * pi * r
LET s=pi * r * r
PRINT “r=”;r,“d=”;d,“s=”;s
END
```

———————————————— ex03-3 ————————————————

```
OK. run
? 5
r=5             d=31.4159            s=78.5398
OK.
```

程序中的 INPUT 语句未选用提示信息，执行时，先显示问号“?”，然后暂停。当操作者键入一个数值并按回车键，系统将把该数值传送到变量 r 对应的的存储单元中，程序继续往下执行，计算 d 和 s 的值，并输出显示。

〔**例3-4**〕 计算若干个梯形的面积

设 a,b,h 分别为梯形的上底，下底和高，s 为梯形的面积。

其程序为：

```
! calculate area of trapezoid
INPUT prompt “a is :”:a
INPUT prompt “b is :”:b
INPUT prompt “h is :”:h
LET s=(a+b) * h/2
PRINT “area=”;s
```

```
END
```

ex03-4

```
OK. run
a is :6
b is :7
h is :8
area= 52
OK. run
a is :3
b is :4
h is :5
area= 17.5
OK.
```

从上可以看出，只要每次运行程序时，操作者键入不同的 a，b，h 的值，就可得到相应的 s 值，而源程序不用改变。若在一个程序中使用了多个 INPUT 语句时，选用提示信息是必要的，否则，每出现一个“?”后，操作者就难于迅速作出反应，该为哪个变量键入数据。

INPUT 语句不仅提高了程序的通用性，也增强了程序对用户的友好程度，用户借助提示信息，可以心平气和地与计算机“对话”。使程序具有良好的用户界面，也正是软件工程中所要求的。

使用 INPUT 语句的几点说明：

(1) 执行 INPUT 语句时，从键盘键入的数据只能是常数或字符串，不能是函数，变量或表达式。

(2) 键入字符串时，如果字符串首尾没有空格，中间没有逗号或惊叹号，则字符串两端可以不用双引号，否则两端必须有双引号。

(3) 在执行一个 INPUT 语句时，如果键入的数据个数少于语句中变量的个数，且按了回车键，屏幕上则显示出如下提示：

```
Too few input items, Please add more.
```

并在下行再次显示问号“?”，等待用户键入尚缺的数据项。

如果键入的数据个数多于语句中变量的个数时，则显示出：

```
Too many input items, Excess ignored.
```

系统将自动舍弃多余的输入数据。

(4) 若需要键入的数据，一行内容纳不下时，可在本行结尾处键入逗号并按回车键，下一行显示出问号“?”后，继续键入其余数据。

第四节　READ/DATA(读/置数)语句与 RESTORE(恢复数据区指针)语句

一、READ/DATA 语句

在 True BASIC 中，除了可用 LET 语句和 INPUT 语句为变量输入数据之外，也可通过配

合使用 READ 语句和 DATA 语句，使变量获得数据。如果一个程序中要对多个变量提供确定值时，若使用 LET 语句，则程序的语句行大量增加，若使用 INPUT 语句，因人机对话，等待用户输入数据，效率很低，而采用 READ 语句和 DATA 语句则是较为合适的方法。

READ 语句称为读数语句，DATA 语句称为置数语句，其格式分别为：

```
READ 〈变量名1〉,〈变量名2〉,…
```

```
DATA 〈常量1〉,〈常量2〉,…
```

DATA 语句是非执行语句，其作用是在程序的执行语句执行前，系统把各 DATA 语句中的全部数据依书写的先后顺序放置于内存的数据区里，为 READ 语句读取数据作准备。

READ 语句是执行语句，其作用是从数据区中读取数据，传送给相应的变量。

为了便于表示从数据区内读取数据的情况，我们可以设想在数据区内有一个“数据指针”。开始读时，“数据指针”指向区内第一个数据，每读取一个数据并传送给相应变量之后，此“指针”自动后移，指向下一个数据。

使用 READ 语句和 DATA 语句需要注意以下几点：

(1) READ 语句和 DATA 语句必须配合使用才起作用，即在程序中若使用了 READ 语句，就一定要使用 DATA 语句来提供数据。

(2) DATA 语句提供的数据与 READ 语句中的变量在类型上必须一一对应。否则，将产生“wrong type”的错误。

(3) DATA 语句中的数据只能为常量，不能为变量，函数或表达式。如果为字符串，且首尾有空格或中间有逗号或惊叹号，其两端必须用双引号，否则，双引号可以省略。

(4) 鉴于 DATA 语句是非执行语句，所以 DATA 语句可放在程序中 END 语句之前的任何位置。

(5) 用 READ 语句读取数据时，如果 DATA 语句提供的数据个数不够，运行程序时将发生“Reading past end of data”(读出界)的错误。

〔**例3-5**〕 鸡兔三十六，共有一百足，问鸡兔各几何？

设鸡兔总头数为 h，总脚数为 f，鸡数量为 x，兔数量为 y。

可列出方程：$\begin{cases} x+y=h \\ 2x+4y=f \end{cases}$

当代计算机还不能直接解方程，必须确定解方程的算法，即求方程根的步骤，利用代入消元法得到

$$\begin{cases} x = \dfrac{4h-f}{2} \\[2mm] y = \dfrac{f-2h}{2} \end{cases}$$

其中 h,f 为已知数据，需要由用户输入，x 和 y 为所求结果，即输出。解题的算法为：

s1：输入鸡兔的总头数 h 和总脚数 f；

s2：按上式计算鸡的数量 x，兔的数量 y；

s3：输出 x，y 的值

实现此算法的程序为：

```
! calculate number of cock and rabbit
READ h,f
LET x=(4*h-f)/2
LET y=(f-2*h)/2
PRINT "cock=";x, "rabbit=";y
DATA 36,100
END
```

```
──────────────────────────── ex03-5 ────────────────────
OK. run
cock=   22            rabbit=   14
OK.
```

〔**例3-6**〕 一个理工类考生考了七门课，求考生的总成绩。

分析：由 DATA 语句提供一个考生的准考证号，姓名，以及七门课的成绩，然后用 READ 语句读到各变量中，再计算总分，最后输出。其程序为：

```
! calculate total score of student
READ number$ ,name$ ,s1,s2,s3,s4,s5,s6,s7
LET total=s1+s2+s3+s4+s5+s6+s7
PRINT number$ ,name$ ,total
DATA "203506","Wang Ping"
DATA 85,74,92,88,76,65,84
END
```

```
──────────────────────────── ex03-6 ────────────────────
OK. run
203506         Wang Ping          564
OK.
```

二、MORE DATA 与 END DATA 函数

为了有效地读取内存数据区中的数据，防止“读出界”的错误，True BASIC 提供了两个数据测试函数，MORE DATA 和 END DATA，函数值为逻辑值。

MORE DATA 函数的功能：测试数据区内是否还有数据可读，若有，函数值为真，否则为假。

END DATA 函数的功能：测试数据区内是否已没有数据可读，若没有，函数值为真，否则为假。

利用数据测试函数，上述计算考生总分的程序可改写为：

```
! calculate total score of student
READ number$ ,name$
LET total=0
```

```
DO while more data
   READ s
   LET total=total+s
LOOP
PRINT number$,name$,total
DATA "203506","Wan Ping"
DATA 85,74,92,88,76,65,84
END
```

———————————————————— ex03-06 ————————————————

```
OK. run
203506          Wan Ping          564
OK.
```

程序中使用了 DO-LOOP 循环结构，当数据区内有数据可读时，执行 DO 语句和 LOOP 语句之间的语句，若无数据可读，则执行 LOOP 的后继语句。关于 DO-LOOP 循环结构将在第五章中详述。

三、RESTORE 语句

使用 READ 语句从数据区读取一批数据，“数据指针”将移到这批数据的后面，若需要把这一批数据再传送给另外一批变量，应使指针重新依次扫过这批数据。但是在执行 READ 语句时，指针只会自动后移，而不会自动前移。可以将这批数据在 DATA 语句中重写一遍。例如：

```
READ A,B,C,D,E,F
READ I,J,K,L,M,N
       ⋮
       ⋮
DATA 1,2,3,4,5,6
DATA 1,2,3,4,5,6
```

到时，指针能够依次扫过重写的数据，将其提供给另外的变量。

如果这一批数据批量很大，且要重复使用多遍，将导致程序冗长和数据区占用的内存空间增大。为了消除这个弊端，可利用 True BASIC 提供的 RESTORE 语句。语句格式为：

```
RESTORE
```

语句的作用是强迫数据指针由当前位置恢复到数据区的开始位置，即指向程序中的第一个 DATA 语句的第一个数据，指针又可以依次指向已读过的数据，将它们提供给另外的变量。

使用 RESTORE 语句，上述的例子可修改为：

```
READ A,B,C,D,E,F
RESTORE
READ I,J,K,L,M,N
  ⋮
```

```
⋮
DATA 1,2,3,4,5,6
```

可见，程序变得简练，且不增大数据区所占用的内存空间。这对于多次重复使用大批量的数据是非常有用的。

至此，已经介绍了 LET，INPUT，READ 三种为变量提供数据的语句。在程序设计中，究意选用哪一种较为合适，应具体问题具体分析。

一般来讲，变量个数不多，且具有已知值或经过计算确定的值时，应使用 LET 语句；若变量个数较多，并都具有已知值，采用 READ 语句与 DATA 语句更为合适；如果需要编制适应各种参数变化的通用程序，变量的值不能在程序中固定，则采用 INPUT 语句。

程序执行时，True BASIC 系统对程序中所有的变量作初始化处理，数值变量初始化为零，字符串变量初始化为空串。如果一个变量从未用任何输入语句提供过数据，则保留初始化值。

第五节　REM(注释)语句和 END(结束)语句

一、REM 语句

在程序中适当地加以注释，可使程序清晰易读。True BASIC 中，程序的注释是通过 REM 语句实现的，其语句格式为：

```
REM        注释字符串
```

其中，关键字 REM 可用惊叹号！代替，注释字符串可以包含 True BASIC 所允许的任何字符，且两端可以省略双引号。

注释语句是非执行语句，运行程序时，系统不对注释内容作语法检查，只是在列程序清单时照原样列出，有无注释语句对程序的执行速度和执行结果不产生任何影响。此语句的作用仅是对整个程序或程序中某一部分进行备注说明，以提高程序的可读性，帮助人们更好地阅读和理解程序。为此，注释应该准确、简单明了。

注释可分为序言性注释和功能性注释两类。序言性注释常置于程序或模块的开头，一般用来说明整个程序或模块的功能，使用的主要算法，以及提供设计人员名单，设计日期等。功能性注释常嵌在程序内部，用来说明一个语句或一个程序段的功能等。

需要说明的是，若要对程序中某一语句进行注释，可在该语句行的末尾置一注释语句，这个注释语句只能以惊叹号！开头，而不能以 REM 开头。

〔例3-7〕 累乘程序

```
REM program of re-multiply
LET p=1                    ! provide initial value
DO
        INPUT x
        IF x=0 then EXIT DO
        LET p=p * x
LOOP until x=0
```

```
PRINT “p=”;p                    ! output result
END
```

这个程序的第一行为序言性注释，其他两处注释为功能性注释。

二、END 语句

END 语句也称为结束语句。一个可直接运行的 True BASIC 程序，在末尾处必须有且只能有一个 END 语句，如果没有 END 语句，运行时将出现“Missing end Statment”的语法错误。END 语句格式为：

```
END
```

机器执行到此语句时，便停止程序的运行。

第六节　程序设计初步

在这一章里，我们已经学习了 True BASIC 中的输入，输出，注释和结束语句，利用这些语句，可以编制一些简单的程序。从现在起，我们要练习编制程序，开如进行程序设计的训练。

下面通过一个具体的例子，说明 True BASIC 程序设计的一般步骤及有关问题。

〔**例3-8**〕 已知三角形的三条边长，求三角形的面积。

1. 分析问题，明确目标和要求

此题简单，要求明确。当问题复杂时，必须对问题进行分析，讨论，并书写详细的任务说明书。

2. 选择合适的数据结构，构造数学模型并确定算法

本题求解属于数学问题，程序中需输入的数据是三角形的三条边长，经抽象后可用数值变量 a，b，c 来表示。根据数学知识，设 $s=(a+b+c)/2$，三角形的面积 area 可表示为：

$$\text{area} = \sqrt{s(s-a)(s-b)(s-c)}$$

这即为求解本题的数学模型，计算出的面积为程序的输出。

数学模型确定之后，就可以考虑用什么方法来实现，即确定算法。本题的算法可表示如下：

s1：输入三条边长 a，b，c；

s2：计算中间结果，即 $(a+b+c)/2 \Rightarrow s$；

s3：按数学模型，计算出三角形面积 area；

s4：输出 area 的值。

3. 算法的图形描述

利用 PAD 图来描述，上述算法的表示如图3-1所示。

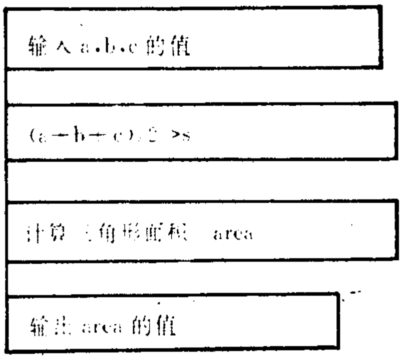


图3-1　求三角形面积的 PAD 图

4. 用 True BASIC 语言实现所描述的算法

接下来，将上面已描述的算法转换为 True BASIC 程序，即进入编码阶段。具体地讲，就是针对图中的每一个框选取合适的语句来实现。

(1) 考虑程序的通用性，用 INPUT 语句输入三条边的值。

(2) 用 LET 语句，实现中间结果 s 的计算。

(3) 计算面积 area，选用 LET 语句。

(4) 用 PRINT 语句，输出 area 的值。

程序末尾，要使用 END 语句。另外，为提高程序的可读性，使用了注释语句对程序功能进行了说明。

程序如下：

```
! calculate area of triangle
INPUT prompt “Enter a,b,c=”： a,b,c
LET s=(a+b+c)/2
LET area=sqr(s*(s-a)*(s-b)*(s-c))
PRINT “area=”； area
END
```

5. 调试和运行程序

程序写完之后，首先应当进行静态检查、验证。即检查书写的关键字，变量名，数据，运算符号，标点符号等是否有错，接着按程序语句执行的顺序“走”一遍，看看按设计要求是否“走”得通，如有错误，书面改正。

程序经静态检查修改，并认定无错后，可将其输入到计算机中，进行动态调试。此时，可使用 RUN 命令，在 True BASIC 解释方式下查错纠错，直到输出运行结果。

对输出的运行结果进行分析。本题的运行结果是输出一个三角形的面积，其值为正数才有实际意义，而上述程序无法保证这一点，因为对 a，b，c 的取值未作限制，输入的数据可能会造成表达式 s*(s-a)*(s-b)*(s-c)值小于等于零，等于零，输出面积为零，小于零，负数不能开平方而出现运行错误。待学习了第四章中的条件语句之后，对程序作适当修改，就能防止这类错误发生。

经分析之后，如果程序运行结果正确，用户可根据需要，利用 True BASIC 提供的编译和汇集功能，将程序转换成编译方式下和操作系统状态下可执行的文件，今后再使用此程序时，可提高运行效率。

习　题

3—1. 简述 LET 语句中的赋值号“=”与数学中的等号“=”有何不同？

3—2. 指出下列语句或程序段中的错误。

(1) LET　X=5:Y=6

(2) LET　X+Y=12.5

(3) LET　A$=HAPPY

(4) LET　NAME=“Li ming”

(5) INPUT　PROMPT “a,b,c=”;a;b;c

(6) READ　A$,B,C$,D,E

DATA　WAN, 24,35,ZHANG,2/3

3—3. 写出下列程序的运行结果。

(1) LET　X=2

```
    LET   Y=4
    PRINT  X,Y X*Y,X/Y
    PRINT  X+Y,X-Y
    END

(2) LET   A=1
    LET   B=2
    LET   C=3
    PRINT  A;B;C;A*B*C
    PRINT  A^3, B^3,C^3
    END

(3) LET   M=5
    LET   N=2
    PRINT  “M+N=”;M+N,“M-N=”;M-N,
    PRINT  “M*N=”;M*N, “M/N=”;M/N
    END

(4) LET   B$=“FORTRAN”
    LET   C$=“PROGRAM DESIGN”
    PRINT  B$ & C$
    PRINT  B$ & C$[1:7]
    END

(5) READ  A,B,C,D,E
    LET   T=A+B+C+D+E
    LET   V=T/5
    PRINT  “TOTAL=”;T,“AVERAGE=”;V
    DATA  87,75,81,77,92
    END

(6) READ  A, B
    DATA  1, 2, 3
    READ  C, D, E
    RESTORE
    DATA  4, 5, 6, 7
    READ  F, G, H, I, J, K, L, M, N
    DATA  8, 9, 10
    PRINT  A, B, C, D, E, F, G, H
PRINT  I, J, K, L, M, N
END
```

3—4. 编写一个程序，从键盘输入二个正数，分别作为半径和高，求圆锥体的体积。

3—5. 已知 A=25，B=17，试编程序输出以 A，B 为边长的矩形的面积。

3—6. 编一程序，从键盘输入一个学生的学号，姓名，性别，年龄，家庭地址，并将输入内容在一行内显示出来。

3—7. 分别用 LET 语句，INPUT 语句，READ/DATA 语句为变量提供数据的方法，各编一个程序，计算平面坐标系上两点 A(0,1)，B(4,4)之间的距离。计算两点间距离的公式为：

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

3—8. 已知不含有数字0的四位整数 N，试编程序将 N 的前两位数字与后两位数字颠倒构成一个新的四位整数，并显示输出。

3—9. 用 INPUT 语句为变量 numb1，numb2，numb3提供数据，编程序以实现找出它们中最大的数和最小的数。

3—10. 已知字符串“HELLO”，“DO␣FORMAT”，“COMPILE”，“DAYS␣OF␣MONTH=30”，“YEAR”，“28”，“365”

(1) 编程序输出前三个字符串中长度较短者。

(2) 用对子串赋值方法，编程序输出以下字符串：“DAYS␣OF␣YEAR=365”，“DAYS␣OF␣MONTH=28”，“MONTH␣OF␣YEAR”，“YEAR=365”。

第四章　选择结构

前面介绍的一些语句称为简单语句，每一语句完成单一的功能，执行时按语句在程序中出现的先后顺序逐行执行。这种结构称为顺序结构。但是在处理实际问题过程中，常需要根据不同的情况选择不同的处理方案，这就是所谓的“逻辑判断”过程。对于这一过程，仅用顺序结构的语句是无法实现的。因此，程序设计语言中必须包含有实现逻辑判断功能的语句成份。True BASIC 中提供的选择(判断)结构可根据给定的条件，对程序中的语句系列有选择地执行，以解决各种需逻辑判断处理的问题。

选择结构是程序设计中一种十分重要的控制结构。本章将详细介绍 True BASIC 的各种选择结构。

第一节　关系表达式和逻辑表达式

在 True BASIC 程序中，条件通常是用关系表达式或逻辑表达式来表示的。

一、关系运算符与关系表达式

关系运算是指同类型的两个量进行比较(即数值量与数值量相比，字符量与字符量相比)，运算的结果是个逻辑值。逻辑值只有两个：真(true)和假(false)。

True BASIC 中使用的关系运算符如表4-1所示。

表4-1　　True BASIC **关系运算符**

运算符	>	>=或=>	<	<=或=<	=	<>或><
含义	大于	大于等于	小于	小于等于	等于	不等于

用一个关系运算符把两个数值表达式(或两个字符串表达式)连接起来所得到的式子，称为关系表达式，简称关系式。

例如，x>0，score>=80，name $ =“WANG HUA”都是合法的关系式，而-1<x<1，y $ =N 则不是关系式。

关系式的运算顺序为：先计算出关系运算符两边的各表达式的值，然后进行关系运算(或称比较运算)，得到一个逻辑值。如关系式 x>0，当 x 取正数时，则关系式的值为真，取零或负数时，关系式的值为假。True BASIC 中由于没有定义逻辑常量和逻辑变量，因此不能在程序中直接输出逻辑值或把逻辑值赋给一个变量。

数值型数据，是按其代数值的大小进行比较的。对字符型数据，则是按字符对应的 ASCII 码值的大小进行比较的。两个字符之间的比较，就是比较它们各自对应的 ASCII 码值，ASCII 码值大的字符为大，反之为小。两个字符串之间的比较，则是从各自左边的第一个字符开始，自左至右，相对应的字符逐一进行比较，如果所有的对应字符都相同，则两个字符串相等，否则为

不等。如果两个字符串不等，其中最先出现的不相同的对应字符的比较结果，即作为两个字符串的比较结果，其后面的对应字符不再进行比较。例如："ABC"和"AXY"进行比较，其中最先出现的不相同的对应字符为"B"和"X"，由于"B"<"X"，所以就可以确定"ABC"<"AXY"，而不要再比较后面的对应字符"C"和"Y"。如果两个字符串长度不等，但从左端开始的所有对应字符都相同，则字符个数较多的字符串为大，如"Your">"You"。

需要指出，在比较两个字符串时，其中的空格也要参与比较，如"A␣B"与"ABC"，其中最先出现的不相同对应字符为"␣"和"B"，由于"␣"<"B"，所以"A␣B"<"ABC"。

根据上述比较规则，下列关系式的值均为真。

"ab">"AB"
"125"<"35"
"BOOK"<"PEN"
"125">"12"
"－5">"＋6"
"ABC␣DEF"<"ABCDEF"

二、逻辑运算符与逻辑表达式

一个关系式只能表示一个简单的条件，但处理实际问题时，有时需要对多个条件进行判断才能决定是否做某件事。比如，某地外语类考生的总分不低于500分，外语成绩不低于85分时才可录取外语专业。这里某个考生是否被外语专业录取，取决于对其总分大于等于500分和外语成绩大于等于85分这两个条件的判断，只有这两个条件都满足时才录取。又如，某班同学凡是三门主课中有一门或一门以上低于80分者，则没有享受奖学金的资格。这里决定一个同学是否享受奖学金，要对每门主课成绩是否小于80分进行判断，即三个条件中只要有一个条件满足就失去了资格。我们把由两个或两个以上的简单条件构成的条件称为复合条件。为了表示复合条件，True BASIC 中提供了逻辑运算符和逻辑表达式。

1. 逻辑运算符

True BASIC 提供了三种逻辑运算符 AND、OR、NOT。

AND，称为逻辑与，用来连接两个逻辑值。当这两个逻辑值均为真时，运算结果为真，否则为假。

OR，称为逻辑或，用来连接两个逻辑值。当这两个逻辑值中只要有一个为真时，运算结果为真，否则为假。

NOT，称为逻辑非，后面置一个逻辑值。当该逻辑值为真时，运算结果为假；当该逻辑值为假时，运算结果为真。

设 A，B 为两个逻辑量，上述三种逻辑运算的含义如表4-2所示。

表4-2　　逻辑运算真值表

A	B	A AND B	A OR B	NOT A	NOT B
真	真	真	真	假	假
真	假	假	真	假	真
假	真	假	真	真	假
假	假	假	假	真	真

2. 逻辑表达式及其运算规则

用逻辑运算符把若干个关系式或逻辑函数连接起来所得到的式子称为逻辑表达式，其运算结果为一个逻辑值。

一个关系式或一个逻辑函数可看作为最简单的逻辑表达式。

例如， x＞－1 AND x＜1
　　P＄＝“N” OR P＄＝“n”
　　NOT（end data OR flag＝1）

均为合法的逻辑表达式。

由于逻辑表达式含有关系式，而关系式可含有数值表达式或字符串表达式。那么在逻辑表达式中就可能同时含有算术运算符，字符串连接运算符，关系运算符和逻辑运算符，还可能含有括号和函数，那么逻辑表达式内按什么规则进行运算呢？True BASIC 规定，其运算顺序为：

括号→函数→算术运算，字符串连接运算→关系运算→逻辑运算

其中算术运算的规则要遵循数值表达式中的规定，即乘方优先，其次为乘，除，最后为加，减，同级运算，自左至右进行。

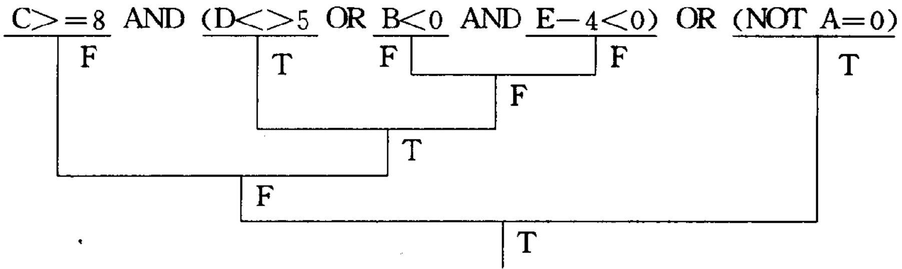
逻辑运算的规则为：逻辑非优先，其次为逻辑与，最后为逻辑或，即 NOT→AND→OR。

〔**例4-1**〕 设 A＝1，B＝0，C＝3，D＝4，E＝5，求以下逻辑表达式的值。

C＞＝8 AND (D＜＞5 OR B＜0 AND E－4＜0) OR (NOT A＝0)

按照前面所述的运算规则，得到该逻辑表达式的值为真。

假如用 T 表示真，F 表示假，其运算过程示意如下：



引进了逻辑表达式之后，复合条件的表示就很方便了，例如：

设总分为 Sum，外语成绩为 foreign，总分大于等于500分且外语成绩大于等于85分可表示为：

sum＞＝500　AND　foreign＞＝85

设 Score1，Score2，Score3分别为三门主课的成绩，若有一门低于80分可表示为：

Score1＜80 OR Score 2＜80 OR Score3＜80

第二节　IF 型选择结构

True BASIC 提供了两种类型的选择结构，一种是 IF 型选择结构，一种是 SELECT CASE 型选择结构。本节将介绍 IF 型选择结构。

IF 型选择结构具有三种不同的语句格式，简单型（IF-THEN），二择一型（IF-THEN-ELSE）和多择一型（ELSE IF），其中前两种格式又可分为单行结构和块结构两种。用户可根据算法要求，灵活地选取 IF 语句，用以表达在算法中出现的各种分支处理、真伪判别等问题。

一、简单 IF 选择语句

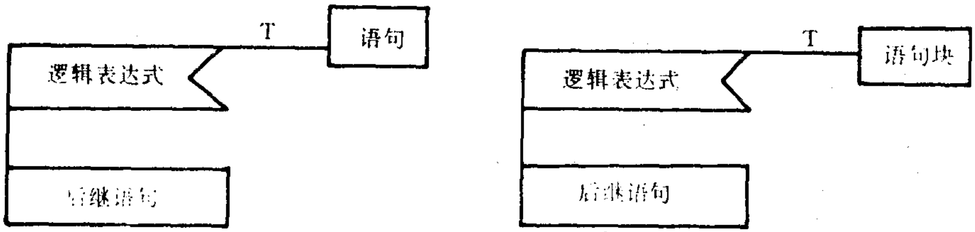
格式1：

```
        IF 〈逻辑表达式〉 THEN 〈语句〉
```

格式2：

```
        IF 〈逻辑表达式〉 THEN
            ……┐
            ……├（语句块）
            ……┘
       END  IF
```

简单型 IF 语句的执行过程是：先计算逻辑表达式的值，若为真，则执行 THEN 后面的语句或语句块；若为假，则直接执行该 IF 语句的后继语句。图4-1的 a 和 b 分别表示了格式1和格式2的执行过程。



(a) (b)

图4-1 简单型(IF-THEN)选择结构执行过程 PAD 图

其中格式1描述的 IF 语句称为单行简单 IF 语句，THEN 后的语句必须为一个可执行语句，且整个 IF 语句必须在一行内书写；格式2描述的语句称为多行(或块结构)简单 IF 语句，语句块可由一个或多个可执行语句构成，IF-THEN 为首行，END IF 为尾行。

〔**例4-2**〕 由键盘输入三角形三条边 a，b，c 的值，如果任意两边之和均大于第三边，则可构成一个三角形，计算并输出该三角形的面积，否则显示出错信息“enter error！”。

```
! calculate area of triangle
INPUT prompt "Enter a, b, c=:":a,b,c
LET flag=0
IF a+b>c and a+c>b and b+c>a then
   LET flag=1
   LET s=(a+b+c)/2
   LET area=sqr(s*(s-a)*(s-b)*(s-c))
   PRINT "triangle area=";area
END IF
IF flag=0 then PRINT "enter error !"
END
```

```
——————————————————————————— ex04-2 ———————————————————————————
OK. run
Enter a, b, c= :7,8,9
triangle area=26.8328
OK. run
Enter a,b,c= :1,2,3
enter error !
OK.
```

该程序中先后使用了简单 IF 选择语句的格式2和格式1，当输入的 a,b,c 的值无法构成一个三角形时，程序显示出错信息。这就避免了因 a,b,c 取值不当造成三角形面积为零或负数开平方的错误。

二、IF-THEN-ELSE(二择一)语句

格式1：

```
    IF 〈逻辑表达式〉 THEN 〈语句1〉 ELSE 〈语句2〉
```

格式2：

```
    IF 〈逻辑表达式〉 THEN
        ……
        ……  }(语句块1)
        ……
   ELSE
        ……
        ……  }(语句块2)
        ……
   END  IF
```

其中格式一为单行二择一型 IF 语句，必须在一行内写完，格式2为多行二择一型 IF 语句。执行二择一 IF 型语句时，先计算逻辑表达式的值，若为真，则执行语句1或语句块1，否则，执行语句2或语句块2。然后执行该 IF 语句的后继语句。其执行过程如图4-2所示。

〔**例4-3**〕从键盘上输入 a,b 两个数，如果 a 大于 b，输出 a－b 的值，否则输出 b－a 的值。

```
! use of if-then-else sentence
INPUT a,b
```

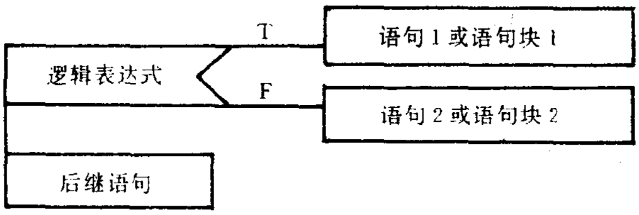


图4-2 二择一型(IF-THEN-ELSE)选择结构执行过程 PAD 图

```
IF a>b then PRINT a－b else PRINT b－a
END
```

〔**例4-4**〕 任意输入20个非零的数，分别计算出正负数之和及其个数。

```
! Program of accumulation and count
LET s1,s2,k1,k2=0
FOR i=1 to 20
      INPUT prompt "x=":x
      IF x>0 then
         LET s1=s1+x
         LET k1=k1+1
      ELSE
         LET s2=s2+x
         LET k2=k2+1
      END IF
NEXT i
PRINT "number of positive"; k1
PRINT "sum of positive"; s1
PRINT "number of negative"; k2
PRINT "sum of negativc"; s2
END
```

程序中的 FOR-NEXT 为 True BASIC 提供的另一种循环结构，用于实现 FOR 与 NEXT 之间语句系列的重复执行。其内容将在第五章详述。

三、ELSE IF(多择一)语句

True BASIC 为解决更为复杂的判断问题。提供了 ELSE IF(多择一)语句。其一般格式为：

```
  IF 〈逻辑表达式1〉 THEN
          ……
          ……  }(语句块1)
          ……
ELSE  IF 〈逻辑表达式2〉 THEN
          ……
          ……  }(语句块2)
          ……
           ⋮
ELSE  IF 〈逻辑表达式 n〉 THEN
          ……
          ……  }(语句块 n)
          ……
ELSE
          ……
          ……  }(语句块 n+1)
          ……
END  IF
```

其中所有语句块均为若干个可执行语句，ELSE IF 部分可以使用任意多个，另外，END IF 前的 ELSE 部分是可选的。

该语句执行时是按从上到下的顺序，依次检查判断各逻辑表达式的值。如果某个逻辑表达式的值为真，则执行该逻辑表达式之下相应的那个语句块，然后结束判断，执行后继语句（即END IF 后面的语句）；如果没有一个逻辑表达式的值为真，则执行 ELSE 后面的语句块，再执行后继语句，若未选用 ELSE 部分，就直接执行后继语句。其执行过程如图4-3所示。

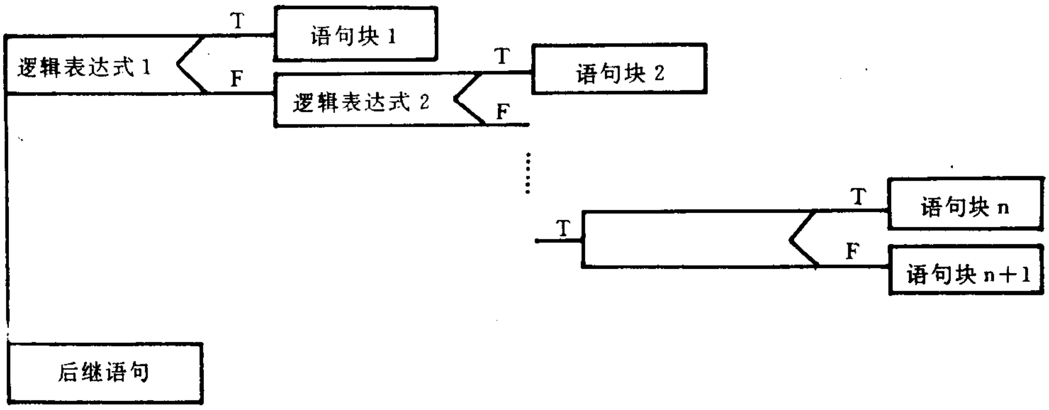


图4-3　多择一型（ELSE IF）选择结构执行过程 PAD 图

〔**例4-5**〕　n 个考生参加了某门课的统考之后，分类统计各分数段的人数。

假如按下述要求分类：

A 等（优）：90分或90分以上

B 等（良）：80～89分

C 等（中）：70～79分

D 等（及格）：60～69分

E 等（不及格）：0～59分

相应的程序为：

```
! calculate tstrength of diffenent score
INPUT prompt "Enter number of student:": n
FOR i=1 to n
    INPUT prompt "Enter a score:": score
    IF score<60 then
       LET e=e+1
    ELSE IF score<70 then
       LET d=d+1
    ELSE IF score<80 then
       LET c=c+1
    ELSE IF score<90 then
       LET b=b+1
    ELSE
       LET a=a+1
    END IF
NEXT i
PRINT "A=";a
PRINT "B=";b
PRINT "C=";c
PRINT "D=";d
PRINT "E=";e
END
```

〔**例4-6**〕 求一元二次方程 $ax^2+bx+c=0$的实根，程序为：

```
! solve a quadratic equation
INPUT prompt "Enter a,b,c=":a,b,c
LET d=b * b－4 * a * c
IF a=0 then
   PRINT "one root at:";－c/b
ELSE IF d>0 then
         LET s=sqr(d)
         PRINT "x1=";(－b+s)/(2 * a)
         PRINT "x2=";(－b－s)/(2 * a)
ELSE IF d=0 then
         PRINT "x1=x2=";－b/(2 * a)
ELSE
         PRINT "no real root !"
END IF
END
```

第三节 SELECT CASE 型选择结构

在 True BASIC 中，还提供了另一种多择一结构，即分情况选择，它是通过使用 SELECT CASE 语句来实现的。其语句格式为：

```
SELECT  CASE 〈表达式〉
          CASE 〈条件11〉, 条件12〉,……
             ……┐
             ……├(语句块1)
             ……┘
        CASE 〈条件21〉, 条件22〉,……
             ……┐
             ……├(语句块2)
             ……┘
             ⋮
        CASE 〈条件 n1〉,〈条件 n2〉,……
             ……┐
             ……├(语句块 n)
             ……┘
        CASE  ELSE
             ……┐
             ……├(语句块 n+1)
             ……┘
END  SELECT
```

其中，表达式（也称为选择表达式）可为数值表达式或字符串表达式，各语句块为若干个可执行语句。

语句执行时，先计算表达式的值，然后按自上而下，自左至右的顺序依次对各CASE后面所列出的条件相比较，如找到一个与表达式值相匹配的条件，就执行其下面相应的语句块，并停止比较，转去执行后继语句（即END SELECT之后的语句）；如找不到相匹配的条件，就执行CASE ELSE后面的语句块，再执行后继语句。上述执行过程如图4-4所示。

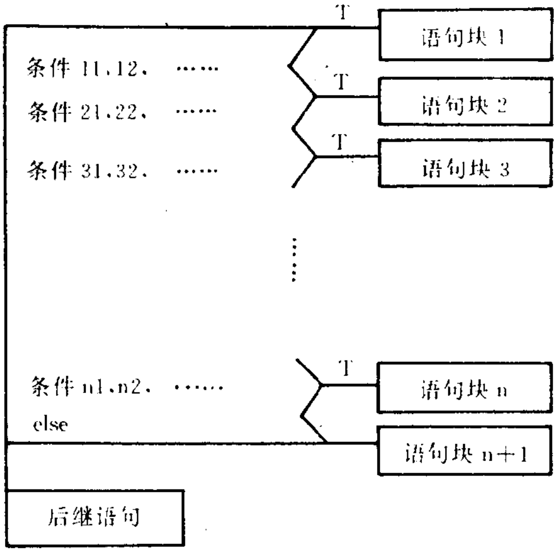


图4-4 分情况（SELECT CASE）选择结构执行过程PAD图

使用SELECT CASE语句需要注意以下几点：

（1）语句中的选择表达式，其类型必须与各CASE后的条件类型相一致。当选择表达式为数值表达式时，各条件均为数值量，为字符串表达式时，各条件均为字符量。

（2）分情况选择中的条件与IF型选择中的条件表示形式不同。分情况选择的条件中只能出现常量，不能出现变量或表达式，条件的表示可为下列三种形式之一或这三种形式的有效组合。

① 常量表

例如： CASE 3，4，5

CASE “BOOK”，“PEN”

当表达式的值为常量表中一个元素时，表示与条件匹配。

② 〈常量1〉 TO 〈常量2〉

例如： CASE 0 TO 59

CASE “A” TO “D”

当表达式的值在此条件区间内，则表示与条件匹配。

③ IS 〈关系运算符〉常量

例如： CASE IS>85

CASE IS <>“WORKER”

当表达式的值能使关系运算符表达的关系成立时，则表示与条件匹配。

上述三种形式可以在一个CASE行中综合应用，实现有效的组合。例如：

```
CASE 45,60 TO 69, IS>80
```

(3) 语句中的 CASE ELSE 部分是可选的。当选择表达式不管取什么值，都能找到一个相匹配的条件时，则可以不选用 CASE ELSE 部分。除此之外，应当选用 CASE ELSE 部分。因为一旦表达式的取值与所有列出的条件均不匹配，结构中又无 CASE ELSE 部分时，则会出现“NO CASE Selected, but no CASE ELSE”的错误信息。可见，选用 CASE ELSE 部分，对提高程序的坚固性是有益的。

〔**例4-7**〕 将例4-5的程序改用 SELECT CASE 分情况选择结构来实现。其程序为：

```
! calculate strength of different score
INPUT prompt “Enter number of student:”: n
FOR i=1 to n
    INPUT prompt “Enter a score:”: score
    SELECT CASE score
    CASE is>=90
         LET a=a+1
    CASE is>=80
         LET b=b+1
    CASE 70 to 79
         LET c=c+1
    CASE 60 to 69
         LET d=d+1
    CASE else
         LET e=e+1
    END SELECT
NEXT i
PRINT “A=”;a
PRINT “B=”;b
PRINT “C=”;c
PRINT “D=”;d
PRINT “E=”;e
END
```

此程序中选用了二种不同方式表示条件，说明条件的表示是可以灵活选用的。

从这个例子还可以看出，SELECT CASE 选择结构与 ELSE IF 选择结构功能类似，都可用来实现多择一分支处理。在解决对一个表达式的不同取值而进行不同处理的某些问题中，它们可以互相替代。

第四节　选择结构应用举例

选择结构是程序设计中经常采用的一种控制结构，True BASIC 提供了 IF 型和 SELECT CASE 型两种选择结构，其中 IF 型选择结构中又有三种不同格式。在具体应用中，究竟选用什

么类型，何种格式，并没有什么固定模式，可由程序设计人员视具体情况，自行选定。下面举一些采用选择结构进行程序设计的例子，以期读者能举一反三，对选择结构运用灵活自如。

〔**例4-8**〕 利用枚举法，找出1～1000之间所有7的倍数的数。

所谓7的倍数就是能被7整除的数，如果一个数 i 满足条件 i/7＝INT(i/7)，则 i 就是7的倍数。本例利用 FOR 循环，遍历1～1000范围内的所有数，逐一按条件判断，符合条件的数则输出。

程序为：

```
! find multiple of seven
FOR i=1 to 1000
    IF int(i/7)=i/7 then PRINT i,
NEXT i
END
```

另外，一个数能被7整除，其余数为0，可将上述程序中的判断条件改为 MOD(i,7)＝0，运行结果相同。

〔**例4-9**〕 求

$$y=\begin{cases}\dfrac{1}{\sqrt{1-x^2}} & \text{当}\ -1<x<1\ \text{时}\\ 2x & x\leqslant -1\ \text{或}\ x\geqslant 1\ \text{时}\end{cases}$$

从题目给出的算式看出，当 x 取不同值时，可按两种不同的式子计算 y 的值。这可以用二择一结构来实现，其算法为：

s1：输入 x 的值

s2：选择不同算式计算 y 的值

s3：输出 x，y 的值

程序为：

```
! calculate y
INPUT prompt “x=”;x
IF x>-1 and x<1 then
   LET y=1/sqr(1-x*x)
ELSE
   LET y=2*x
END IF
PRINT “x=”;x, “y=”;y
END
```

也可以将程序中的判断条件改为 x<=1 or x>=1，程序相应改为：

```
! calculate y
INPUT prompt “x=”;x
IF x<=-1 or x>=1 then
   LET y=2*x
ELSE
```

```
    LET y=1/sqr(1-x*x)
END IF
PRINT "x=";x,"y=";y
END
```

〔**例4-10**〕 任意输入三个彼此不等的数 a,b,c,要求按从大到小的顺序输出这三个数。

算法设计：

s1：输出三个彼此不等的数；

s2：同时比较三个数是不可能的，一次只能比较两个数。先比较 a,b 的大小，其中较大者放入 m 中，较小者放入 n 中；

s3：拿 c 与 m 比，若 c>m，则顺序输出 c,m,n；

s4：若 c<m，再拿 c 与 n 比，如 c>n，则顺序输出 m,c,n，如 c<n，则顺序输出 m,n,c。

该算法的 PAD 图如图4-5所示。

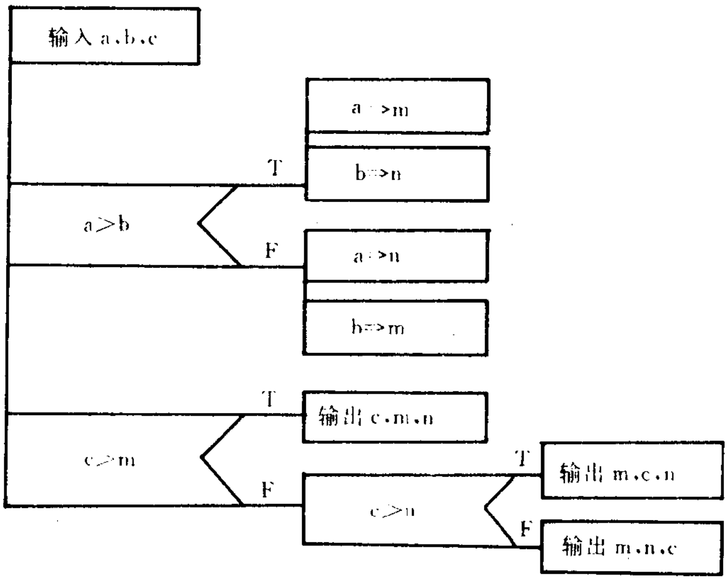

输入 a,b,c
a=>m
b=>n
T
a>b
F
a=>n
b=>m
T
输出 c,m,n
c>m
F
T
输出 m,c,n
c>n
F
输出 m,n,c

图4-5 任意三个数按从大到小顺序输出的 PAD 图

其程序为：

```
! print three number in regular sequence
INPUT prompt "Enter a,b,c:": a, b, c
IF a>b then
    LET m=a
    LET n=b
ELSE
    LET m=b
    LET n=a
END IF
IF c>m then
```

```
   PRINT c,m,n
ELSE IF c>n then
   PRINT m,c,n
ELSE
   PRINT m,n,c
END IF
END
```

〔**例4-11**〕 按下述规定编写计税程序

（1）货价≤2000元，免税

（2）2000元<货价≤5000元，纳税2%

（3）货价>5000元时，5000元以内部分纳税2%，超过5000元的部分纳税5%。

算法设计：

s1：输入货价 price；

s2：根据 price 值，分三种情况，算出纳税额 tax；

s3：输出 price，tax。

其程序为：

```
! calculate tax
INPUT prompt "price=":price
SELECT CASE price
CASE is<=2000
       LET tax=0
CASE 2000.01 to 5000
       LET tax=price * 0.02
CASE else
       LET tax=5000 * 0.02+(price-5000) * 0.05
END SELECT
PRINT "price=";price, "tax=";tax
END
```

〔**例4-12**〕 从键盘输入年号，判断该年是否为闰年。年号符合下列条件之一的是闰年：

（1）能被4整除，但不能被100整除

（2）能被100整除，且能被400整除

如是闰年，显示年号和“leap year”，否则显示年号和“common year”。

程序为：

```
! determine a leap year
INPUT prompt "year=":year
PRINT year;"————";
IF yar/100<>int(year/100) then
   IF year/4=int(year/4) then
      PRINT "leap year"
   ELSE
      PRINT “common year”
   END IF
ELSE
   IF (year/400)=int(year/400) then
      PRINT “leap year”
   ELSE
       PRINT “common year”
   END IF
END IF
END
```

IF 型语句是一种执行语句，也可以出现在 IF 型语句的语句块中，构成条件语句嵌套。此时，程序中将会出现多个 THEN 与 ELSE，True BASIC 规定，每个 ELSE 总是与前面最邻近一个未配对的 THEN 相匹配。

需要指出，在使用 IF 嵌套结构时，要做到完全的嵌套，在使用多行 IF 语句时，必须保证 IF 与 END IF 完全配对，如果分层较多，需注意不要丢失应有的 END IF。

判断某年是否为闰年的问题除了可用上述 IF 嵌套结构来解决外，也可以用 ELSE IF 多择一选择结构来实现，相应程序为：

```
! determine a leap year
INPUT prompt “year=”:year
PRINT year;“———————”;
IF year/4<>int(year/4) then
   PRINT “common year”
ELSE if year/100<>int(year/100) then
    PRINT “leap year”
ELSE if (year/400)=int(year/400) then
    PRINT “leap year”
    ELSE
PRINT “common year”
END IF
END
```

习　题

4—1. 把下面的命题用逻辑表达式表示出来：

(1) A 小于 B 且小于 C。

(2) A 和 B 皆大于 C。

(3) 三角形的三条边 a，b，c，必须满足任何两边的和大于第三边。

(4) A 或 B 中有一个等于 C。

(5) 两门课的成绩都在90分以上或者两门课总分不低于185分。

4—2. 设 A=2,B=3,C=1,D=0,试求下列逻辑表达式的值:

(1) A+B>3 OR D<0 AND (C<>0 OR C+D * A ^ 3>=7)

(2) A-D=0 OR (NOT B>0) AND (A-B)/C<>0

4—3. 指出下列语句中的错误,并改正。

```
(1) IF A=B$  THEN PRINT A,B$
(2) IF R > 3 THEN   X=PI * R * R   ELSE   X=2 * PI * R
(3) IF X < 8
        THEN   LET X=SQR(X)
        ELSE   LET X=X ^ (1/3)
    END
(4) SELECT  CASE  X
      CASE  2,3
        PRINT  X
      CASE  “A”,“B”
        PRINT  X$
      CASE  ELSE
        PRINT  “END”
      END  SELECT
(5) LET  B=6
    SELECT  CASE  Y
      CASE  IS=B
        PRINT  B * B
      CASE  IS<B
        PRINT  B ^ 3
      END  SELECT
```

4—4. 写出下列程序的运行结果。

```
(1) LET x=14
    IF  X>0  THEN  PRINT  SQR(x)
    PRINT  “END”
    END
(2) LET  A=8
    LET  B=5
    IF  A>B  THEN
        PRINT  A-B
    ELSE
        PRINT  A+B
    END  IF
    END
```

```
(3) LET  S=76
    IF  S>=90  THEN
        PRINT  "SCORE:  A"
    ELSE IF  S>=80 THEN
        PRINT  "SCORE:  B"
    ELSE IF  S>=70 THEN
        PRINT  "SCORE:  C"
    ELSE IF  S>=60 THEN
        PRINT  "SCORE:  D"
    ELSE
        PRINT  "SCORE:  E"
    END  IF
    END
(4) LET  X=-5
    SELECT  CASE  X
      CASE  IS>0
        PRINT  "POSITIVE"
      CASE  IS=0
        PRINT  "ZERO"
      CASE  ELSE
        PRINT  "NEGATIVE"
      END  SELECT
    END
```

4—5. 根据题意，分别编出相应的程序。

(1) 任意输入三个长度不等的字符串，要求按其长度由长到短的顺序输出这三个字符串。

(2) 任意输入三个数，找出其中的最大数及最小数。

(3) 已知 $Z=\begin{cases} \dfrac{1}{xy} & \text{当 } x\neq0\text{且 } y\neq0\text{时} \\ x+y & \text{当 } x=0\text{或 } y=\text{时} \end{cases}$

从键盘输入 x，y 的值，计算并输出 Z 的值

(4) 给出数据 n 和 x 后，求 f(x)的函数值，其中 n 为1～5的整数。

$$f(x)=\begin{cases} x^2+1 & n=1 \\ \sqrt{x+1} & n=2 \\ L_nx & n=3 \\ 2x & n=4 \\ |x| & n=5 \end{cases}$$

(5) 若银行存款的月利率是：一年期7.2‰，二年期7.8‰，三年期8.4‰，五年期9.6‰，八年期11.4‰，输入本金 x 元，分别计算到期时，可获得本金与利息之和

各为多少元?

(6) 某单位规定一月内全勤者发100%奖金，请假1～2天者，发80%奖金，请假3～4天者，发50奖金，请假5天或5天以上者，不发奖金，输入全勤奖金额Q(元)和一个职工的请假天数D，求其所得奖金数J。

第五章　循环结构

在解决实际问题中，经常要进行某些连续的有规律的重复运算或操作。因此，在程序设计中就需要将某个程序段连续重复执行多次，具有这种功能的结构称为循环结构，它是程序设计中又一种重要的控制结构。进入和退出循环往往受一定条件控制，按照不同的控制方式，True BASIC 提供了两种不同形式的循环结构，即 FOR 循环结构和 DO 循环结构。并且还提供了 EXIT FOR 语句和 EXIT DO 语句，以满足从中途跳出相应循环的特殊需要。

第一节　FOR 循环和 EXIT FOR 语句

一、FOR 循环

FOR 循环又称计数循环，其循环次数通常是预定的。

1. FOR 循环的一般格式

```
FOR var=e1   TO   e2   STEP   e3
      ……  ┐
      ……  ├(语句块)
      ……  ┘
NEXT var
```

其中，var 为循环变量，它只能为数值型简单变量，在相匹配的 FOR 语句与 NEXT 语句中循环变量名必须一致。e1，e2，e3均为数值表达式，分别为循环变量的初始值，终限值和增量。循环变量的增量也称为步长，当步长为1时，“STEP 1”可缺省。FOR 语句与 NEXT 语句之间的语句块为循环体，表示要重复执行的程序段，通常由若干个可执行语句构成。预定执行循环体的次数由表达式 e1，e2，e3的值确定。

一个 FOR 循环结构由 FOR 语句，循环体，NEXT 语句三部分构成，缺一不可。整个 FOR 循环结构的功能是按预定次数执行循环体。

2. FOR 循环的执行过程

(1) 计算 e1，e2，e3各表达式的值，并存储起来；

(2) 把初始值赋给循环变量 var；

(3) 将循环变量 var 的值与终限值进行比较，即检测 var 值是否超过 e2的值，若超过，循环过程结束，转去执行 NEXT 语句的后继语句；

(4) 若 var 值未超过 e2值时，顺序执行循环体内各语句；

(5) 执行 NEXT 语句，将循环变量 var 的值加一步长 e3的值，然后转去执行第(3)步。

上述执行过程可用图5-1来表示。

FOR 循环执行过程也可用图5-2所示的 PAD 图进行描述。

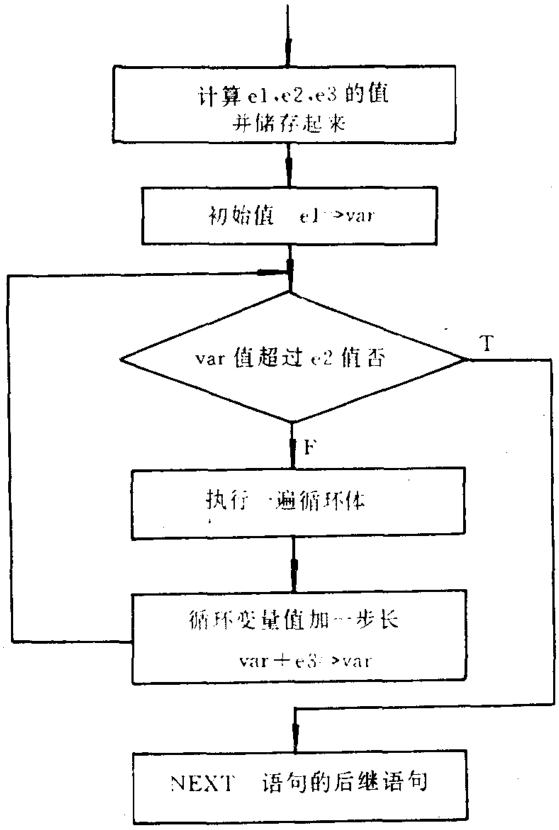

计算 e1、e2、e3 的值
并储存起来
初始值 e1→var
var 值超过 e2 值否
T
F
执行一遍循环体
循环变量值加一步长
var+e3→var
NEXT 语句的后继语句

图5-1　FOR 循环执行过程流程图

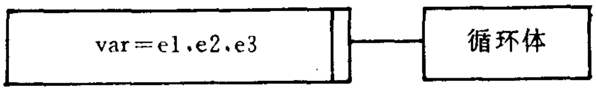

var=e1,e2,e3
循环体

图5-2　FOR 循环执行过程 PAD 图

3. 几点说明

(1) 在进入 FOR 循环时,e1,e2,e3的值就被确定并存储起来,即使在循环体中对它们重新赋值,也不改变预定的循环次数。

(2) 步长值可正,可负,但不可为零。当步长值为正时,循环变量取值由小到大递增变化,每当取值小于或等于终限值时,执行一遍循环体,当取值大于终限值时,谓之“超过”,退出循环。换言之,步长值为正时,循环的终止条件为循环变量的值大于终限值。

〔**例5-1**〕

```
! positive step
FOR i=1 to 10 step 2
    PRINT i; “     ”;
NEXT i
END
```

ex05-1

```
OK. run
1           3           5           7           9
OK.
```

当步长值为负时，循环变量的取值由大到小递减变化，每当取值大于或等于终限值时，执行一遍循环体，当取值小于终限值时，谓之“超过”，退出循环。换言之，步长值为负时，循环的终止条件为循环变量的值小于终限值。

〔**例5-2**〕

```
! negative step
FOR i=10 to 1 step -3
     PRINT i;"      ";
NEXT i
END
```

ex05-2

```
OK. run
10            7            4            1
OK.
```

如果步长值为零，循环变量取值总是等于初始值（初始值超过终限值的特殊情况除外），永远不会超过终限值。这样，将会无休止地重复执行循环体，使程序运行陷入“死循环”状态，这是应该避免的。若一旦出现“死循环”，可使用 Ctrl+Break 键中断程序的运行。

(3) 循环变量不一定能取到终限值（见例5-1），不论是否取到了终限值，在正常结束循环时，其取值一定超过了终限值（利用 EXIT FOR 语句退出循环的情况除外）。

(4) FOR 循环结构预定执行循环体的次数为

$$\mathrm{INT}((e2 - e1)/e3) + 1$$

如果在循环体内对循环变量重新赋值，则会改变预定的循环次数。因此，一般不提倡在循环体内进行改变循环变量取值的操作。

〔**例5-3**〕

```
FOR i=1 to 10 step 2
     PRINT i,
     LET i=i * 2
NEXT i
END
```

ex05-3

```
OK. run
1              4              10
OK.
```

在上述程序中，由于在循环体内使用了赋值语句 LET i=i * 2，对循环变量 i 重新赋值，结

果使循环变量取值和循环次数发生了变化。

(5) 在循环体内可以出现循环变量，也可以不出现循环变量。若不出现，循环变量仅起控制循环次数的作用。

〔**例5-4**〕

```
FOR day=1 to 7
    READ week $
    PRINT ucase $ (week $ [1:1] & lcase $ (week $ [2:3]);"     ";
NEXT day
DATA monday, tuesday, wednesday, thursday
DATA friday, saturday, sunday
END
```

```
———————————————————— ex05-4 ————————————————————
OK. run
Mon      Tue      Wed      Thu      Fri      Sat      Sun
OK.
```

若将上述程序中的 FOR 语句改为

```
FOR day=11  to 17
```

运行结果不变。

下面是利用 FOR 循环结构进行程序设计的几个例子。

〔**例5-5**〕国际象棋发明者古代印度宰相达依尔受到国王奖赏，国王问他需要什么奖品，他说："在国际象棋的64个格子中，第1格放一粒麦子，第2格放2粒，第3格放4粒，以后每格的麦粒数是前一格的两倍，一直放到第64格，我就满意了。"试编程序，计算共需多少立方米麦子(1立方米小麦约有1.4亿粒)。

这个问题的数学模型为：$total=2^0+2^1+\cdots\cdots+2^{63}$

算法分析，这是一个典型的利用迭代算法求累加的问题。由于各项具有明显的规律，第 n 项(n=1,2,……,64)可书写为2∧(n-1)，因此可利用迭代法构造算法，其迭代算式为：total+2∧(n-1)⟹total；迭代变量 total 的初始值为零，迭代次数为64。首先将 total 置零，然后利用 FOR 循环，重复执行迭代算式64次，即可求得累加和 total 的值。

算法设计为：

s1：累加变量 total 置初值0；

s2：构造一个 FOR 循环，循环变量 n 依次取值1,2,……,64，每取一次值，执行一次累加操作：total+2∧(n-1)⟹total；

s3：将 total 值转换成立方米数；

s4：输出 total 的值。

程序如下：

```
! calculate quantity of wheat
LET total=0
FOR n=1 to 64
```

```
    LET total=total+2^(n-1)
NEXT n
LET total=total/1.4e+8
PRINT "total=";total
END
```

——————————————————ex05-5——————————————————

```
OK. run
total=1.31762e+11
OK.
```

这相当于全世界几百年的粮食总产量。

在累加前，对累加变量置零，是一种好习惯，否则，如果累加变量中原有一个非零的值，就会影响累加结果的正确性。

〔**例5-6**〕 计算 n!

第一章已讲述，计算阶乘可利用递推法构造算法。若计算 n！，可以从初始条件 f(0)=1出发，应用递推公式 f(i)=i*f(i-1)逐步求出 f(1),f(2),f(3),……,最后求出 f(n)的值。

算法设计为：

```
s1：输入任一正整数 n；
s2：累乘变量 f 置初值1；
s3：构造一个 FOR 循环，循环变量 i 取值从1变到 n，每获得一个值，执行递推运算：f * i ⟹ f；
s4：输出 f 的值。
```

程序如下：

```
! calculate factorial of the number n
INPUT prompt "Enter a number:";n
LET f=1
FOR i=1 to n
    LET f=f*i
NEXT i
PRINT n;"!=";f
END
```

——————————————————ex05-6——————————————————

```
OK. run
Enter a number:4
 4     != 24
OK. run
Enter a number:6
 6     != 720
```

OK.

累乘前，累乘变量置1是必须的，因为系统将对任何一个数值型变量初始化为零，所以若省略置1操作，就达不到预定的累乘目的。

〔**例5-7**〕 用梯形法求定积分 $\int_a^b \sin x dx$

求 $\int_a^b \sin x dx$ 在区间[a,b]上的定积分（见图5-3），其几何意义是求 $y=\sin(x)$ 和 $y=0$ 及 $x=a$，$x=b$ 所围成的图形的面积。

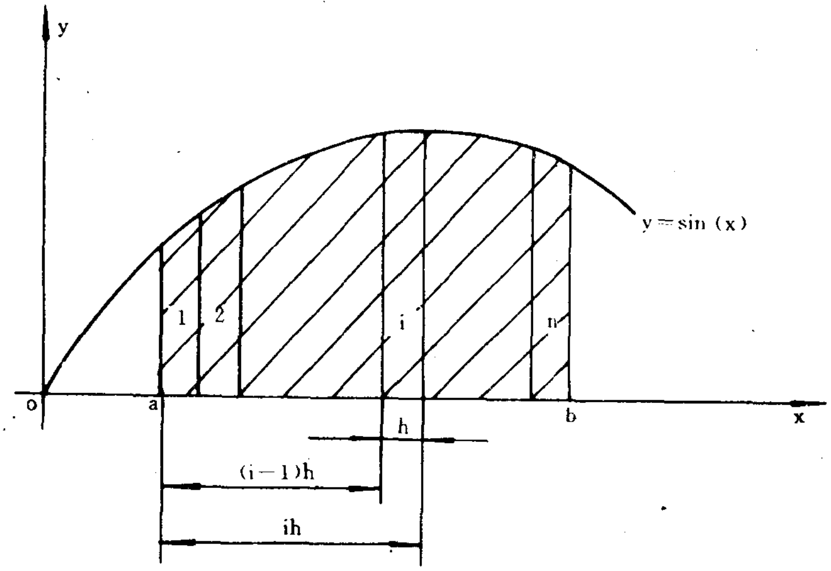


图5-3 $\int_b^a \sin x dx$ 的几何意义

欲求出图中阴影部分的面积，可把它沿 x 轴分成 n 等份，每一等份就是一个小的曲边梯形。n 的取值越大，求得的面积值愈接近于实际值。

计算梯形面积的公式为((上底+下底)＊高)/2。对于本题，每个梯形的高 h 均为(b+a)/n，第 i 个梯形的上底为 sin(a+(i−1)＊h)，下底为 sin(a+i＊h)，其面积为(sin(a+(i−1)＊h)+sin(a+i＊h))＊h/2。

由分析可以看出，计算定积分的问题已转换为 n 个曲边梯形面积的累加问题。为此，可构造出以下算法：

s1：输入 a，b，n 的值；

s2：计算出梯形的高 h；

s3：累加变量 s 置初始值0；

s4：构造一个 FOR 循环结构，循环变量 i 取值从1到 n，i 每获取一个值，即执行一次迭代算式 s+(sin(a+(i−1)＊h)+sin(a+i＊h))＊h/2⟹s；

s5：输出 a，b，n，s 的值

程序为：

```
! trapezoidal method
INPUT prompt "a,b,n=":a,b,n
```

```
LET h=(b-a)/n
LET s=0
FOR i=1 to n
    LET s=s+(sin(a+i-1)*h)+sin(a+i*h))*h/2
NEXT i
PRINT “a=”;a,“b=”;b,“n=”;n,“s=”;s
END
```

```
——————————————————————————————ex05-7——————————————————
OK. run
a,b,n=1,2,50
a=1             b=2             n=50              s=.956417
OK. run
a,b,n=1,2,100
a=1             b=2             n=100             s=.956441
OK.
```

〔**例5-8**〕有时，某些重要的数据出于保密的目的，需要以密码的形式存放，这就需要编制密码生成程序和译码程序。密码生成，也称编码，就是将一个字符串按一定的规则转换成另一个新的字符串。读取加密数据需要作译码处理，译码是编码的反变换过程，即将密码数据按规则重新转换为原来的数据。

编码的方法和规则有很多种。假如我们采用这样一种规则：空格字符(其 ASCII 码值为32)保持不变，其他字符，将其 ASCII 码值加10后，利用 CHR $ 函数，转换成新的字符。由于 CHR $ 函数的自变量 n 的最大取值为255，如果原 ASCII 码值加10后超过255，可使用 MOD(n,256)的值作为自变量。例如字符“A”的 ASCII 码值为65，加上10为75，CHR $ (75)返回字符“K”，即将“A”编码为“K”。译码则是将“K”反变换为“A”。

下面给出的程序实现一个字符串的编码和译码运算。鉴于一个字符串中每个字符均需要按上述规则进行变换，因此，利用 FOR 循环实现这一处理过程，循环体重复执行的次数应与字符串的长度相等。

程序为：

```
! codify code
PRINT “enter origiral code”;
INPUT orig $
LET new $ =“”
FOR k=1 to len(orig $ )
    LET t $ =orig $ [k:k]
    LET n=ord(t $ )
    IF n<>32 then
       LET n=n+10
       IF n>=255 then LET n=mod(n,256)
    END IF
    LET new $ =new $ & chr $ (n)
NEXT k
PRINT "the new code is:"; new $
! decord
LET orig $ =""
FOR k=1 to len(new $ )
    LET t $ =new $ [k:k]
    LET n=ord(t $ )
    IF n<> 32 then
       LET n=n-10
       IF n<0 then LET n=n+256
    END IF
    LET orig $  =orig $ & chr $ (n)
NEXT k
PRINT "the decoded messageis:"; orig $
END
```

ex05-8

```
OK. run
enter origiral code? This is a book.
the new code is: ^ rs} s} k lyyu8
the decoded message: This is a book.
OK.
```

二、EXIT FOR 语句

程序设计中，有时需要在循环变量尚未超过终限值时就跳出循环。True BASIC 提供的 EXIT FOR 语句可实现这一功能，其语句格式为：

```
EXIT  FOR
```

该语句的作用是从 FOR 循环结构的循环体内跳出，转去执行 NEXT 语句的后继语句。

需要指出，EXIT FOR 语句只能出现在 FOR 循环结构的循环体内，而不允许出现在其他地方。EXIT FOR 通常与选择结构配合使用，提供一种满足条件就跳出循环的手段。显然，FOR 循环结构存在着两种结束循环的情况，一种是执行预定的循环次数，正常结束，一种是通过执行 EXIT FOR 语句，强制性退出循环，即不执行完预定的循环次数，提前结束循环。因此，EXIT FOR 语句使 FOR 循环结构增强了灵活性，从而方便算法的描述，并能提高程序的执行效率。

〔**例5-9**〕 试编制通用程序，求全班（不超过50人）某门课程的平均成绩。

算法分析：平均成绩等于总成绩除以总人数。为编制对人数不等的各班都适用的通用程

序，可构造一个 FOR 循环，预定执行50次循环体。若全班人数小于50，则在输入该班所有同学的考试成绩之后，输入一个特殊的数据－1，作为成绩输入的结束标志。设 score 为学生成绩，sum 为总成绩，n 为学生人数，aver 为平均成绩。循环体内，每输入一个学生的成绩，检测是否为－1，若不是，则执行一次 Sum＋Score⟹Sum 和 n－1⟹n 的操作；若是，表明全班所有成绩处理完毕，跳出循环。然后计算出全班的平均成绩并输出。

其程序为：

```
! calculate average of scores
LET n、sum=0
FOR i=1 to 50
    INPUT prompt "Enter a score:": score
    IF score=-1 then EXIT FOR
    LET n=n+1
    LET sum=sum+score
NEXT i
LET aver=sum/n
PRINT "average=";aver
END
```

第二节　DO 循环和 EXIT DO 语句

一、DO 循环

DO 循环是 True BASIC 提供的另一种循环控制结构。由于在 DO 循环中可以方便灵活地使用当型条件（WHILE〈条件〉）和直到型条件（UNTIL〈条件〉）进行循环控制，常用来构造那些难于预定重复执行遍数的循环。DO 循环中的控制条件可为任意逻辑表达式或逻辑函数，下面分别介绍用当型条件和直到型条件控制的 DO 循环。

1. 当型（WHILE）DO 循环

根据条件出现的位置不同，其格式有以下两种：

```
DO  WHILE 〈条件〉
        ……
        ……  }（循环体）
        ……
  LOOP
```

```
DO
        ……
        ……  }（循环体）
        ……
LOOP  WHILE〈条件〉
```

相应的执行流程分别如图5-4和图5-5所示：

两种格式的共同点是当条件成立时，执行循环体，条件不成立时，不执行循环体而退出循环；两种格式的区别在于，左边的格式先判断后执行，若开始就遇条件不成立，则循环体一次都不执行，而右边的格式是先执行后判断，即至少要执行一次循环体。

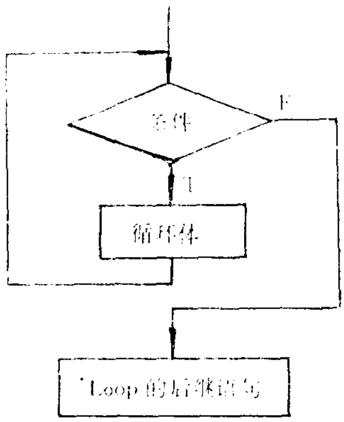

F
条件
T
循环体
Loop 的后继语句

图5-4　DO WHILE-LOOP 结构执行流程图

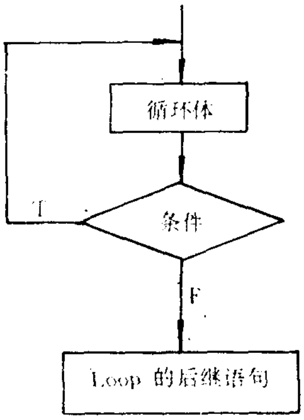

循环体
T
条件
F
Loop 的后继语句

图5-5　DO-LOOP WHILE 结构执行流程图

2. **直到型(UNTIL)DO 循环**

与当型 DO 循环相仿，其格式也有两种：

```
DO  UNTIL  (条件)
      ……
      ……  }(循环体)
      ……
LOOP
```

```
DO
      ……
      ……  }(循环体)
      ……
LOOP  UNTIL  (条件)
```

相应的执行流程分别如图5-6和图5-7所示。

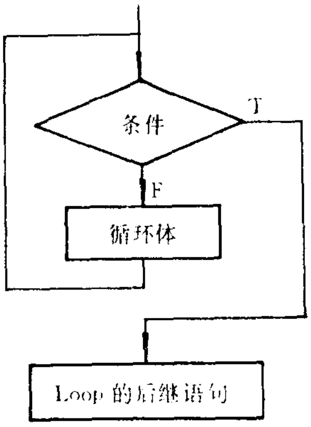

T
条件
F
循环体
Loop 的后继语句

图5-6　DO UNTIL-LOOP 结构执行流程图

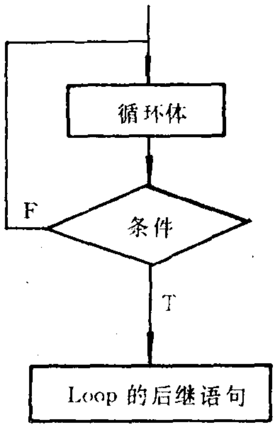

循环体
F
条件
T
Loop 的后继语句

图5-7　DO-LOOP UNTIL 结构执行流程图

这两种格式的共同点是在条件成立时，都不执行循环体而退出循环，条件不成立时，则要执行循环体；其区别在于，左边的格式先判断后执行，若一开始条件就成立，根本不进入循环，即循环体一次都不执行，而右边的格式为先执行后判断，即至少要执行一次循环体。

DO 循环执行流程也可用 PAD 图来描述，见图5-8所示，其中的(a)，(b)，(c)，(d)分别对应于图5-4，5-5，5-6，5-7。

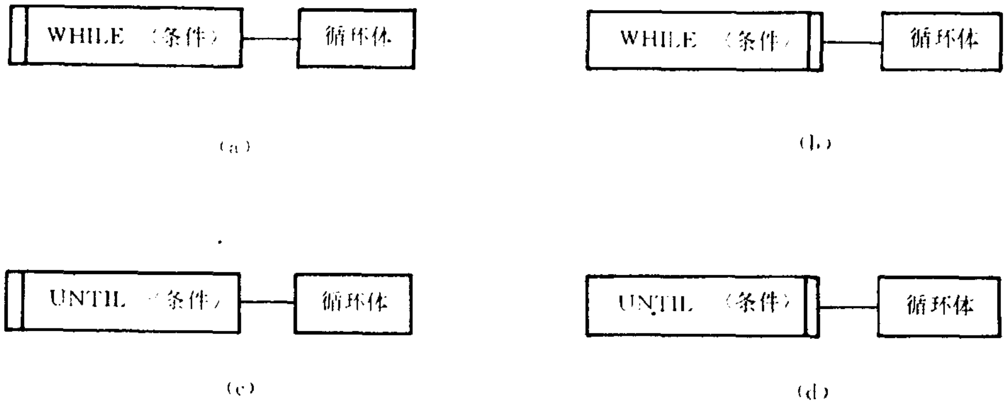


图5-8　DO循环执行过程PAD图

不管是当型，还是直到型控制方式，DO循环结构的功能是根据给定条件重复执行循环体。其执行过程可以归纳为：

(1) 首先检查DO部分有无条件短语，如DO后没有条件短语，则无条件地执行循环体；当DO后有条件短语，WHILE〈条件〉为真时或UNTIL〈条件〉为假时，执行循环体，否则结束循环，执行LOOP的后继语句。

(2) 每执行一次循环体后，检查LOOP后有无条件短语，如LOOP后没有条件短语，则无条件地转到DO语句；如LOOP后有条件短语，当WHILE〈条件〉为真或UNTIL〈条件〉为假时，则控制转到DO语句，否则结束循环，去执行LOOP语句的后继语句。

DO循环结构的循环体一般由若干个可执行语句构成。但需要特别指出的是，在循环体内必须至少有一个改变循环控制条件的语句，以便经有限次重复执行循环体后最终能退出循环，即保证运行程序时不会陷入“死循环”状态。这一点，和FOR循环结构的循环体是有明显区别的。

〔**例5-10**〕 从若干个给定数据中，找出其中的最大值与最小值。

设max放最大值，min放最小值，数据由DATA语句提供。其算法为：

s1：将第一个数提供给max和min；

s2：构造一个DO循环结构，在DO后用more data逻辑函数作为控制条件，其循环体是：

(1) 读取数据指针所指数据，且送入工作单元x中。

(2) 将x值与max值比较，如果x大于max，则x⟹max；

(3) 将x值与min值比较，如果x小于min，则x⟹min；

s3：输出max与min的值。

程序如下：

```
! find max and min of data
READ max
LET min=max
DO while more data
   READ x
   IF x>max then LET max=x
   IF x<min then LET min=x
```

```
LOOP
PRINT "max=";max,"min=";min
DATA -1,3.14,-4.2,6.9,-19,32.8,0.5,1,0.01,-124
END
```

————————————————————————————ex05-10————————————————

```
OK. run
max=32.8          min=-124
OK.
```

如果将上述程序中的DO while more data-LOOP格式改为DO-LOOP until end data格式，其运行结果不变。

〔**例5-11**〕 设一年期的定期存款年利率为8.64%，某人一次存入1000元，以后每年到期时，他将结算的本利和继续以该种形式存入银行，试计算经过多少年后，连本带利比最初的本金翻一番还多？

算法分析：设当年的本金为x，y为计年单元，那么一年后的本利和为x＊(1+0.0864)，再将此作为本金，来计算下一年的本利和，即每过一年，将x＊(1+0.0864)$\Longrightarrow$x，y+1$\Longrightarrow$y重复一次，依此类推，当x$\geqslant$2000时，结束该重复过程。

程序为：

```
! calculate money on bank account
LET x=1000
PRINT "year", "money"
DO until x>=2000
    PRINT y,x
    LET y=y+1
    LET x=x*(1+0.0864)
LOOP
PRINT y,x
END
```

————————————————————————————ex05-11————————————————

```
OK. run
year            money
  0               1000
  1               1086.4
  2               1180.26
  3               1282.24
  4               1393.03
  5               1513.38
  6               1644.14
  7               1786.19
  8               1940.52
```

```
 9            2108.18
OK.
```

将上面程序中的 DO until x>=2000—LOOP 格式改为 DO-LOOP while x<2000，其运行结果相同。

〔**例5-12**〕 编一程序，从给定的一批英文单词中，检索出以字母“A”或“a”，“B”或“b”开头的单词，并统计其个数。

```
! find the word
LET n=0
DO
      READ x$
      IF ucase$(x$[1:1])=“A” or ucase$(x$[1:1])=“B” then
         PRINT x$
         LET n=n+1
      END IF
LOOP until end data
PRINT “n=”;n
DATA PEN,BOOK,aeroplane, bus, bag, pencil
DATA lamp, clock, tea, APPLE, pin, arms
END

———————————————————————ex05-12———————————————
OK. run
BOOK
aeroplane
bus
bag
APPLE
arms
n= 6
OK.
```

二、EXIT DO 语句

EXIT DO 语句和前面所述的 EXIT FOR 语句相类似，它提供一种从 DO 循环内转出，以结束循环的手段。

语句格式为：

```
EXIT  DO
```

语句的作用是从 DO 循环结构的循环体内转出，去执行 LOOP 的后继语句。

EXIT DO 语句只能放在 DO 循环的循环体内，且常和 IF 语句一起使用。如果在循环体内

使用了 EXIT DO 语句，那么在 DO 和 LOOP 后面皆可以缺省条件短语。

〔**例5-13**〕 利用公式 $e=1+\frac{1}{1!}+\frac{1}{2!}+\cdots\cdots+\frac{1}{n!}$ 求数学常数 e 的近似值，要求精确到被舍去的首项小于10^{-6}。

算法分析：这是一个利用迭代法求累加和的问题，解题的关键在于如何计算出除第一项以外其余各项的值，以及如何确定累加的项数。第一项特殊，先把它存入 e 中，以后的 n 项都是有规律的，即第 n 项的值等于其前一项的值除以 n。求这些项的累加和可用循环结构来实现，在循环体内每计算出一项后，与给定的精度值进行比较，如大于精度值，则把该项的值累加到 e 中，并继续计算下一项，当算到某一项小于精度值时，则退出循环，并输出 e 的值。其程序为：

```
! approximatel calculate e
LET e=1
LET t=1
LET n=0
DO
     LET n=n+1
     LET t=t/n
     IF t<1e-6 then EXIT DO
     LET e=e+t
LOOP
PRINT “n=”;n, “e=”;e
END
```

ex05-13

```
OK. run
n=  10          e=  2.71828
OK.
```

第三节 循环嵌套

如果循环结构的循环体内包含有一个或多个循环结构，称为循环嵌套或多重循环。

〔**例5-14**〕 输出算术中的九九乘法表，用 i 和 j 表示被乘数和乘数，这里被乘数 i 取值要从1变到9，需要用一层循环，而对于每一个被乘数，乘数 j 取值也要从1变到9，这又需要一层循环，整个程序要用到二层循环(或称二重循环)，才能达到目的。

其执行流程如图5-9所示。

相应的程序为：

```
! print the multiplication table
FOR i=1 to 9
     FOR j=1 to 9
          PRINT i; “*”;j;“=”;i*j,
    NEXT J
    PRINT
NEXT i
END
```

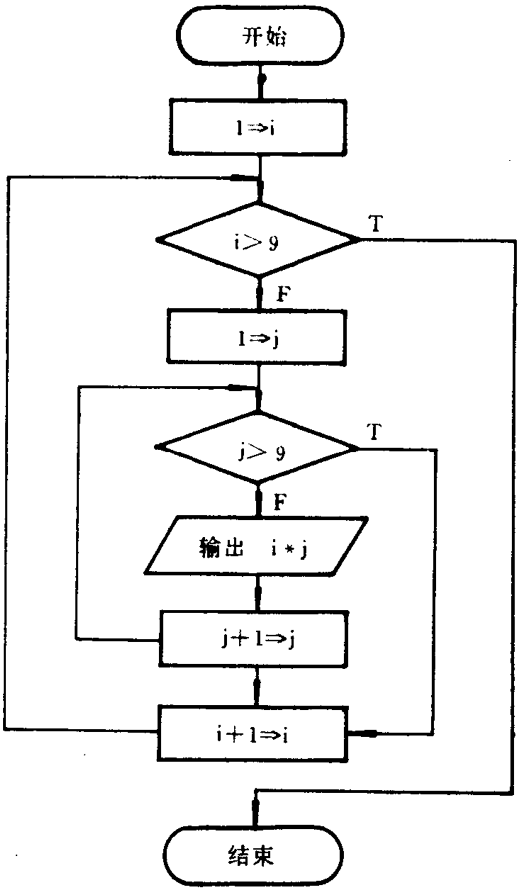

开始
1⇒i
i>9
T
F
1⇒j
j>9
T
F
输出 i*j
j+1⇒j
i+1⇒i
结束

图5-9 输出九九乘法表流程图

这个问题的算法的 PAD 图描述，如图5-10所示。

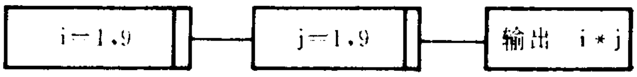

i=1,9
j=1,9
输出 i*j

图5-10 输出九九乘法表 PAD 图

〔**例5-15**〕 计算 $\sum_{n=1}^{20} n!$

这个问题也可用二重循环来实现，内循环实现求一个数的阶乘，外循环控制执行20遍内循环，并把每个算出的阶乘进行累加处理。其执行过程的 PAD 描述如图5-11所示。

其程序为：

```
! calculate s=1!+2!+3!+……+20!
LET s=0
```

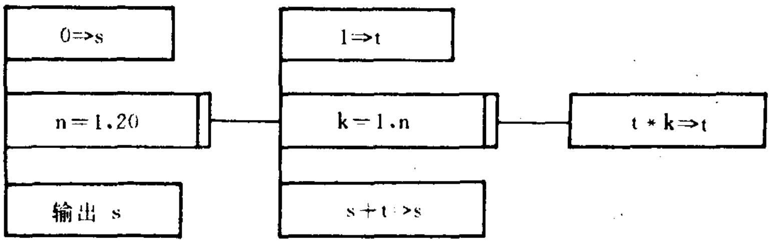


图5-11　求$\sum_{n=1}^{20}n!$的 PAD 图

```
FOR n=1 to 20
    LET t=1
    FOR k=1 to n
        LET t=t*k
    NEXT k
    LET s=s+t
NEXT n
PRINT "n=";s
END
```

ex05-15

```
OK. run
s= 2.56133e+18
OK.
```

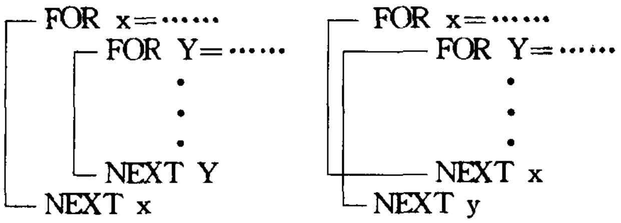
FOR 循环的循环体内，若含有 FOR 循环，这种结构被称为 FOR 循环嵌套。执行时，总是先相对固定外层循环变量的值，而按规律改变内层循环变量的值，待内层循环变量的值超过该层终限值时，外层循环变量的值才改变一次，一直到最外层循环变量的值也超过其终限值时，才结束整个多重循环。

使用 FOR 循环嵌套结构时，需要注意以下两点：

(1) 不同层的循环变量不能同名。

(2) 内外层循环不能交叉，必须完全嵌套。

如：



```
FOR x=……
    FOR Y=……
        ·
        ·
        ·
    NEXT Y
NEXT x
```

正确

```
FOR x=……
    FOR Y=……
        ·
        ·
        ·
    NEXT x
NEXT y
```

错误

如果在 DO 循环结构的循环体内又含有 DO 循环时，则构成了 DO 循环嵌套。在多层 DO 循环嵌套中，每个 LOOP 总是与前面最邻近的尚未配对的一个 DO 相配。

〔**例5-16**〕 1990年第四次全国人口普查结果表明我国人口已达11.6亿，试计算若人口的自然增长率分别为2%，1.5%，1%，0.5%时，到哪一年我国人口将开始超过13亿？

分析题意可知，对四种增长率依次处理构成外循环，其中在每种增长率的情况下，每过一年，年数加1，计算一次 P＊(1＋增长率)，并判断一次 P 是否超过13，(这里 P 为某年人口数)，此重复过程为内循环。

其程序为：

```
! calculate tpopulation of china
DO while more data
   LET p=11.6
   LET y=1990
   READ r
   DO
      LET y=y+1
      LET p=p*(1+r)
      IF p>=13 then EXIT DO
   LOOP
   PRINT "r=";r,"y=";y,"p=";p
LOOP
DATA 0.02,0.015,0.01,0.005
END
```

```
——————————————————————— ex05-16———————————————————————
OK. run
r=  .02          y=  1996          p=  13.0635
r=  .015         y=  1998          p=  13.0673
r=  .01          y=  2002          p=  13.0712
r=  .005         y=  2013          p=  13.01
OK.
```

程序中，也可以使用 FOR 循环结构内包含 DO 循环，或 DO 循环结构内包含 FOR 循环的混合型嵌套形式。

例如，对例5-13求 e 的近似值，精确到被舍去的首项小于10^{-6}，可以用以下程序来实现。

```
! approximately calculate e
LET e=1
LET n=0
DO
   LET n=n+1
   LET t=1
   FOR k=1 to n
      LET t=t/k
```

```
    NEXT k
    IF t<1e-6 then EXIT DO
    LET e=e+t
LOOP
PRINT "n=";n,"e=";e
END
```

ex05-013

```
OK. run
n=   10          e=   2.71828
OK.
```

在书写多层循环嵌套程序时，通常采用缩格形式，使成对的 FOR 和 NEXT，DO 与 LOOP 分层对齐，以使程序清晰易读。

需要说明的是，在多层循环嵌套结构中，若某层内使用了 EXIT FOR 语句或 EXIT DO 语句，其功能只能退出出现该语句的那一层循环。如果需要由内层循环退出外循环，可采用以下的程序结构实现。

```
FOR i=…
    ⋮
    LET flag=0
    FOR j=…
        ⋮
        IF 〈C〉 THEN
           LET flag=1
           EXIT FOR
        END IF
        ⋮
    NEXT j
    IF flag=1 then EXIT FOR
    ⋮
NEXT i
```

上述结构中引入了标志变量 flag，当条件 c 满足即改变 flag 的取值并退出内循环，执行 NEXT j 后的 IF 语句时，因条件成立则退出外循环。

第四节　循环结构应用举例

至此，本书已先后介绍了顺序结构，选择结构和循环结构，它们是结构化程序设计中的三种基本结构。在此基础上，我们可以综合应用所学过的知识，进行范围广泛的程序设计。

一、枚举算法设计

利用枚举法构造解题算法在程序设计中得到广泛使用。枚举法基本思想是，对所要解决的问题的所有可能情况一个不漏地进行检查判断，从中找出符合给定条件的答案来，只有在所有情况检查判断后，尚未找到符合条件的答案时，才能作出“找不到”的结论。枚举算法通常采用循环结构来实现。

〔**例5-17**〕 如果一个三位整数等于其各位数字的立方和，则称这个数为水仙花数，如153 $=1^3+5^3+3^3$，$370=3^3+7^3+0^3$，试编程序找出所有的水仙花数。

分析：可用两种方法列举出全部三位整数。方法之一，从100，101，102，…一直顺序列举到999，对列举的每个数均需检测是否符合水仙花数的条件，即这个数是否等于其三位数字的立方和，如果是，则输出该数。显然，整个列举过程可用单层循环实现。其程序为：

```
! find number of “narcissus flower”
FOR i=100 to 999
     LET a=int(i/100)
     LET b=int(i/10)-a * 10
     LET c=i-a * 100-b * 10
     IF i=a^ 3+b^ 3+c^ 3 then PRINT i
NEXT i
END
```

——————————————————ex05-17——————————————

```
OK. run
 153
 370
 371
 407
OK.
```

方法之二，假定 a，b，c 分别为三位整数的百位，十位，个位数字，变量 a 从1，2，…列举到9，变量 b 和 c 从0，1，…列举到9，采用三层循环嵌套结构，可依次得到100，101，…999的三位数，同时对其进行判断是否为水仙花数，若是，输出该数。相应的程序为：

```
! find number of “narcissus flower”
FOR a=1 to 9
     FOR b=0 to 9
          FOR c=0 to 9
               LET i=100 * a+10 * b+c
               IF i=a^ 3+b^ 3+c^ 3 then PRINT i
          NEXT c
     NEXT b
NEXT a
```

```
END
```

———————————————————ex05-017———————————————

```
OK. run
 153
 370
 371
 407
OK.
```

〔**例5-18**〕 到图书馆查阅某本书，借阅者忘记了书的全名，只记得其中几个连续字符，试编一程序，把含有这几个字符的所有书名显示出来。

算法分析：这实际是解决“是否存在”的问题，列举范围为全部图书，对所列举的每本书需要检测，书名中是否含有某子串（即记得的几个字符），若是，显示该书名，并继续列举，若遍历全部图书之后仍未找到，则给出“未找到”的信息。由于事先并不确定图书数目，因此采用DO循环结构来实现。程序为：

```
! search the book
PRINT “enter your words”
INPUT y$
LET flag=0
DO while more data
   READ bookname$
   LET n=pos(bookname$ , y$ )
   IF n>0 then
      LET flag=1
      PRINT bookname$
   END IF
LOOP
IF flag=0 then PRINT “haven't found!”
DATA “computer”,“computer world”,“data processing”
DATA “software”,“algol”,“true basic”,“fortran77”
DATA “basic program”,“cobol”, “pascal”,“basic language”
DATA “the C programmer's handbook”
END
```

———————————————————ex05-18———————————————

```
OK. run
enter your words? basic
true basic
basic program
basic language
OK. run
enter your words? PL/1
haven't found!
OK.
```

〔**例5-19**〕 三十六块砖,三十六个人搬,成年男搬四,女搬三,两个小孩抬一砖,要求一次全搬完,问需成年男、女及小孩各若干?

设成年男、女及小孩的人数分别为a,b,c,则可以建立此问题的数学模型:

$$\begin{cases} a + b + c = 36 & (1) \\ 4a + 3b + c/2 = 36 & (2) \end{cases}$$

这是一个不定方程求解问题。从方程解的定义出发,可任取一组a,b,c的值代入方程(1)方程(2)作检测,若(1)和(2)都成立,则为一组解,否则不是一组解。

而a,b,c的取值范围可分析得到,如果36块砖全由成年女人和小孩来搬,则成年男人数为0,如果全由成年男人来搬,则男人数为9,所以a为0~9之间的整数。同理,可以推出b为0~12之间的整数,那么小孩数c为36-(a+b)。

构造算法时,a从0~9,b从0~12两层枚举,对于某个a,b值,由方程(1)求出c,用方程(2)来检测,若方程(2)表示的关系成立,则输出a,b,c的值。这一过程可通过双重循环来实现。其程序为:

```
! move bricks
PRINT "man","women","child"
FOR a=0 to 9
     FOR b=12 to 0 step -1
          LET c=36-(a+b)
          IF 4*a+3*b+c/2=36 then PRINT a,b,c
     NEXT b
NEXT a
END
```

———————————————————————— ex05-19 ————————————

```
OK. run
man              woman            child
  3                3                30
OK.
```

二、迭代算法设计

迭代,是构造计算机解题算法的另一种基本方法。它是通过重复执行有关语句,从迭代变量的原值推出它的一个新值,直到满足一定的条件为止。

迭代分为精确迭代与近似迭代。精确迭代是指算法本身事先就确定了迭代次数,并且以此来控制迭代过程,运算结果在理论上应该是精确的,如果有误差,是由于计算机用有限的字长

表示实数造成的，而与算法本身无关。例如求给定的若干个数的累加和及累乘积即为精确迭代。但在解决相当多的实际问题时，找不到求其精确解的算法或者算法难于实现，常利用迭代法求其近似解，这类迭代称为近似迭代。只要迭代过程收敛，迭代次数越多，求得的近似解越接近于精确解。在解决这类问题时，一般无法预定迭代次数，而是给定一个允许误差值，当迭代到近似解的误差范围小于此给定值时，便可以结束这个迭代过程。

迭代算法一般也是通过循环结构来实现。

〔**例5-20**〕 17世纪德国数学家莱布尼兹把圆周率 π 表示成：

$$\frac{\pi}{4}=1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\cdots\cdots+(-1)^{n-1}\cdot\frac{1}{2n-1}+\cdots\cdots$$

试设计一个计算 π 的近似值的 True BASIC 程序，要求被舍去的首项的绝对值小于10^{-3}。

通过变换，$\pi=4-\frac{4}{3}+\frac{4}{5}-\frac{4}{7}+\cdots\cdots+(-1)^{n-1}\cdot\frac{4}{2n-1}$

这里无法预先确定算到哪一项，才会满足给定的条件。

设 n 为迭代变量，n 从0开始，通过执行 LET n=n+1，得到 n 的新值，利用$(-1)^{n-1}\cdot\frac{4}{2n-1}$通项公式算出第 n 项的值，然后将其绝对值与给定的精度值10^{-3}进行比较，如果不小于10^{-3}，则累加到变量 pai 中，一直重复这个过程，直到某项绝对值小于10^{-3}则停止累加，此时变量 pai 的值即为 π 的近似值。

其程序为：

```
! calculate the circumference rate
LET pai=0
LET n=0
DO
    LET n=n+1
    LET x=(-1)^(n-1)*4/(2*n-1)
    IF abs(x)<1e-3 then EXIT DO
    LET pai=pai + x
LOOP
PRINT "pai=";pai
END
```

```
————————————————————————— ex05-20 ——————————————————————
OK. run
pai=   3.14109
OK.
```

〔**例5-21**〕 已知牛顿迭代公式为 $x_{n+1}=x_n-f(x_n)/f'(x_n)$，试利用牛顿迭代公式计算 $\sqrt{a}$ 的近似值，计算精度为10^{-5}，设 a 为1～10范围的自然数。

算法分析：根据题意可列出方程 $x=\sqrt{a}$，即 $f(x)=x^2-a=0$，由牛顿迭代公式 $x=x_0-f(x_0)/f'(x_0)$得到：

$$x=x_0-(x_0^2-a)/(2x_0)=(x_0+a/x_0)/2$$

相邻两次迭代的误差为$-f(x_0)/f'(x_0)$，其绝对值为：

$$|f(x_0)/f'(x_0)| = |(x_0^2 - a)/2x_0| = |(x_0 - a/x_0)/2|$$

当计算出的误差绝对值小于给定的允许误差时，则结束迭代过程。

由于题意中求各数的平方根的允许误差相同，提供的 x 的初值（即第一次给出的近似根 x_0）也可相同，因此可在循环之外提供这两个值。

其程序为：

```
! approximately calculate the square roots
! using the Newton's recurrence formula.
INPUT prompt "error allowend eps:": eps
INPUT prompt "first x0:": x0
FOR a=1 to 10
    DO
        LET error=abs((x0-a/x0)/2)
        LET x=(x0+a/x0)/2
        LET x0=x
    LOOP until error<eps
    PRINT "the squre root of";a; "is;";x0
NEXT a
END
```

ex05-21

```
OK. run
error allowend eps: 1e-5
first x0: 5
the square root of 1 is :  1.
the square root of 2 is :  1.41421
the square root of 3 is :  1.73205
the square root of 4 is :  2
the square root of 5 is :  2.23607
the square root of 6 is :  2.44949
the square root of 7 is :  2.64575
the square root of 8 is :  2.82843
the square root of 9 is :  3.
the square root of 10 is :  3.16228
OK.
```

习　题

5—1. 写出下列程序的执行结果。

```
(1) FOR i=1 to 5
          READ x
          PRINT x,
    NEXT i
    DATA 1,3,5,7,9,11
    END
(2) FOR a=10 to 0 step -3
          PRINT a * a
    NEXT a
    END
(3) LET y$ ="computer"
    FOR i=8 to 1 step -1
          PRINT y$[i;i];
    NEXT i
    END
(4) FOR k=1 to 20
          IF k>15 then EXIT FOR
          PRINT k;
    NEXT k
    END
(5) LET n=0
    LET s=0
    DO while n<=10
        LET n=n+1
        LET s=s+n
    LOOP
    PRINT "s=";s
    END
(6) LET n=0
    LET p=1
    DO
          LET n=n+2
          LET p=p * n
    LOOP until n=8
    PRINT "p=";p
    END
(7) FOR i=1 to 4
          FOR j=1 to 4
                IF i=j then
                PRINT “1”;
            ELSE
                PRINT “0”;
            END IF
        NEXT j
        PRINT
    NEXT i
    END
(8) LET s=0
    DO while more data
        READ x
        LET k=0
        FOR n=1 to x
            LET k=k+n
        NEXT n
        LET s=s+k
    LOOP
    PRINT “s=”;s
    DATA 1,2,3,4,5
    END
(9) LET a$ =“I come from Beinjing”
    LET a$ =a$ & chr$(32)
    LET x=0
    FOR i=1 to len(a$)
        IF ord(a$[i:i])<>32 then
            LET x=x+1
        ELSE
            PRINT a$[i-x:i-1]
            LET x=0
        END IF
    NEXT i
    END
(10) DO until end data
        READ s$
        LET m,n,k=0
        FOR i=1 to len(s$)
            SELECT CASE s$[i:i]
            CASE “A” to “Z”
                LET m=m+1
            CASE “a” to “z”
                 LET n=n+1
            CASE else
                 LET k=k+1
            END SELECT
        MEXT i
        PRINT “m=”;m, “n=”;n,“k=”;k
    LOOP
    DATA Introduction to PASCAL
    DATA Microcomputer & Application
    DATA True BASIC Program
    END
```

5—2. 分别编制程序，计算下列各题中 s 的值。

(1) $s=1^2+2^2+\cdots\cdots+20^2$

(2) $s=1+(1+2)+(1+2+3)+\cdots\cdots+(1+2+3+\cdots\cdots+10)$

(3) $s=1\times(1+2)\times(1+2+3)\times\cdots\cdots\times(1+2+3+\cdots\cdots+10)$

(4) $s=1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\cdots\cdots-\frac{1}{19}$

5—3. 试编程序，找出下列数据中的负数，并且求出这些负数的和。

$-5,-2,-15.34,20,-19.7,162,-23,-1.72$

5—4. 试编程序，挑出上题数据中最大数及最小数。

5—5. 编程序实现，从给定的一批字符串中找出最长的串及序号。

5—6. 有一个数列第一项为0，第二项为1，从第三项起，每项均为它的前两项之和，试编程序输出该数列的前六十项。

5—7. 铁路托运快件行李，规定每张火车票托运的行李重量不超过50公斤时，按某种单价计费，如超过50公斤时，超过部分则按此单价加倍计费。试编一程序，实现按输入的单价和行李重量计算出托运费。

5—8. 某班期中考试了三门课，其中两门为主课，要求挑出满足下列条件之一者：

(1) 两门主课均在95分以上；

(2) 三门课总分在270分以上；

(3) 一门主课为100分，其他两门课均不低于80分。

5—9. 以1980年我国国民经济总产值为基数，计算以年平均增长率分别为5%，6%，7%，8%时，各需要多少年，我国国民经济总产值可以翻两番？

5—10. 求一个四位整数，它满足以下两个条件：

(1) 它的个位数与十位数一样，百位数与千位数一样；

(2) 它的平方根是个两位整数。

5—11. 用牛顿迭代法求方程 $x^3-2x-5=0$ 在 $x_0=2$ 附近的根，精确到 10^{-3}。

5—12. 任意输入两个字符串 Subst \$ 和 st \$ ，若前者是后者的子串，则计算出子串 Subst \$ 在 St \$ 中曾出现的次数，并全部删除之；若前者不是后者的子串，则显示“无法删

除”的信息。

5—13.甲乙两个自然数的和、差、积、商四数加起来等于243，求甲乙两数，若它们的和、差、积、商四数之积为94221，则甲乙两数又各为多少？

第六章　数组与MAT簇语句

迄今我们遇到的变量都是简单变量，其特点是各自独立，相互之间不存在任何联系。然而，在处理实际问题时，常常需要考虑由一些个体数据组成的数据集合。例如：某中学50名教师某学期授课时数（一名教师的授课时数是简单个体，而全体教师的授课时数则为简单个体的集合）；统计100名学生参加四门课程考试的成绩表（表格中每项数据是个体，一行数据表示一名学生的各课成绩，整个表格则是个体数据的集合）等，它们由若干个个体组成，数据之间具有一定的相互关系。

一般地说，数据个体间的结构关系称为数据结构。本章介绍的数组就是一种常用的数据结构。它与简单变量不同之处在于有多个分量，数组的每一个分量（称为数组元素）是一个变量，可以和简单变量一样被赋值和在表达式中使用。

有关数组的重要问题是访问其元素的方法。因此，就读者而言，如何把被处理的对象抽象为数组，以及如何灵活地访问其元素，实现基于数组的各种运算（计算、统计、查询、排序、插入、删除等）是学习本章的核心问题。

本章将介绍数组的定义，数组的输入输出，实现数组特定运算的 MAT 簇语句，以及数组的动态调整等内容，并讨论程序设计中基于数组的各种运算。

第一节　数组及数组元素

一、数组的概念

True BASIC 中，数组是一组按一定顺序排列且具有相同属性的变量的集合。例如，$T=(T_1、T_2、\cdots、T_{50})$就是一个数组，表示50名教师某学期授课时数的集合，又如：

$$\text{score}=\begin{pmatrix} \text{score}_{1,1} & \text{score}_{1,2} & \text{score}_{1,3} & \text{score}_{1,4} \\ \text{score}_{2,1} & \text{score}_{2,2} & \text{score}_{2,3} & \text{score}_{2,4} \\ \vdots & \vdots & \vdots & \vdots \\ \text{score}_{100,1} & \text{score}_{100,2} & \text{score}_{100,3} & \text{score}_{100,4} \end{pmatrix}$$

也是一个数组，表示100名学生参加四门课程考试的成绩的集合。

数组是一种常用的数据结构，与简单变量的不同之处在于有多个分量，每个分量亦称为数组元素。例如，T 数组包括50个数组元素，score 数组包括400个数组元素。

程序设计中，若处理的对象被组织为数组，则应首先为其命名。数组的命名规则与简单变量的命名规则相同，如 T，score 都是合法的数组名。根据数组元素的取值类型，数组分为数值型数组和字符串型数组。与简单变量的类型定义相仿，数组的类型定义体现在数组名后有无类型说明符“＄”。若无“＄”，为数值型数组；若有“＄”，则为字符串型数组。

注意，类型不同的量不能组织成数组。例如，医院对入校新生进行体检，每人有：学号、姓号、性别、年龄、身高…等多项数据，处理这些数据所确定的变量的集合不能是一个数值型数

组，因为姓名、性别是字符串型数据。当然，可以考虑把所有量组织成一个字符串型数组，但是，其中对应于年龄、身高的量不能直接作算术运算，必要时应利用字符串函数将其转换为数值量，然后才能参与算术运算。

二、数组元素的表示方法及数组的维数

True BASIC 规定，数组元素用数组名后跟圆括号括起来的下标来表示。若有多个下标，下标之间用逗号分隔。下标表示该元素在数组中的顺序，或称处于的位置。如“T(42)”即为 T 数组中的一个元素，“T”代表数组名，“42”代表下标。

下标可以是常数、数值型变量(包括数值型数组元素)或数值表达式。下标应当取整数值，否则，系统将自动对其作取整处理。如，T(42)，T(i)，score(5,4)，score(j+1,4)均为正确引用的数组元素，其中 i，j 应事先赋适当的值。

数组元素亦称下标变量，可以和简单变量一样被赋值或在表达式中使用。只有循环变量例外，它只能是简单变量，而不能是下标变量。

下标变量中的下标个数表明该数组的维数。上述 T 数组是一维数组，score 数组是二维数组。二维数组也称为矩阵(Matrix)。True BASIC 允许使用多维数组，但读者应注意，引用二维以上的数组元素时，其下标个数必须与数组的维数相一致。

三、数组的存储结构

数组在计算机中的表示(或称映射)称为数组的存储结构，亦称物理结构。True BASIC 对数组采用连续空间分配，即在内存中开辟一批地址相互连续的存储单元来表示数组。例如，T 数组的存储结构如图6-1所示。

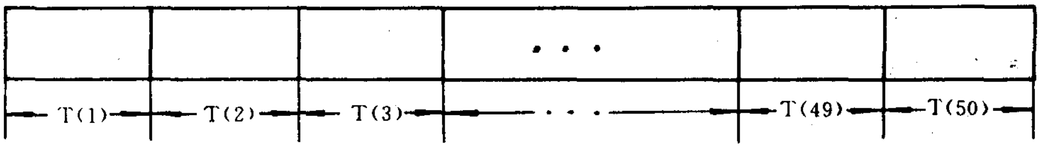


图6-1　T 数组的存储结构

二维数组的存储结构采用“按行分配”的方式，即在内存中先开辟表示第一行各元素的若干个存储单元，接着再开辟表示第二行各元素的存储单元，…。例如，score 数组的存储结构如图6-2所示。

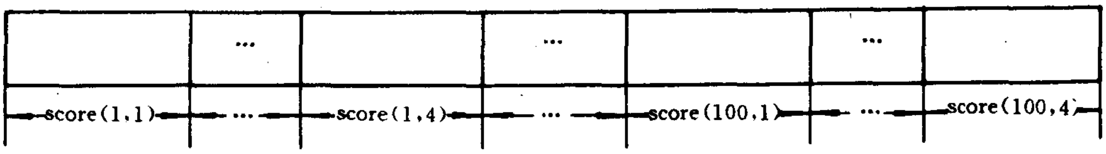


图6-2　score 数组的存储结构

可见，数组的维数是针对数组的逻辑结构而言的，而一个数组不论其维数多少，在存储结构上均呈现为线性顺序排列形式。

第二节　DIM(数组定义)语句和 MAT REDIM(数组重定义)语句

一、DIM 语句

程序中，使用数组元素之前，必须事先用 DIM(DIMension 的缩写，意为定维)语句进行定义。其一般格式为：

```
DIM〈数组1〉(〈下界11〉to〈上界11〉[,〈下界12〉to〈上界12〉]…)[,〈数组2〉(〈下界21〉to〈上界21〉,[,〈下界22〉to〈上界22〉]…)]…
```

功能：该语句将为系统提供：(1)数组名及数组类型；(2) 数组的维数；(3) 数组各维的上下界及各维的大小等信息，并对数组作初始化处理，即将数值型数组各元素置零，字符串型数组各元素置空串。

例　DIM name＄(1 to 100)，score(1 to 100，1 to 4)，year(91 to 94)

共定义了三个数组。其中 name＄是字符串型一维数组，最小下标值取1，最大取到100，共有100个数组元素。score 被定义为数值型二维数组，共有100行4列，400个元素。year 也是数值型一维数组，含有4个数组元素：year(91)，year(92)，year(93)，year(94)。

定义数组时，应注意以下事项：

(1) 数组一旦定义，数组名不得再用作简单变量名。

(2) 所有的数组都应当先定义，后使用。数组的引用分两种形式：数组元素引用和整体引用。前者作为一个变量来使用，其作用与简单变量相仿，后者则以数组名的形式(不能带下标)通过 MAT 簇语句实现数组的整体输入、输出、赋值，以及两数组求和、差、积等特定的运算。

(3) 数组某维下标的上下界，是指该维下标的取值范围。例如，year 数组的最小和最大下标值分别为91和94。若程序中引用数组元素时，其下标值超出该维的上下界时，将会显示“下标超界”的出错信息。例如，以 year(95)的形式引用 year 数组的元素则是错误的。

(4) 定义数组时，下标的上下界应该是整常数，而不能使用变量或数值表达式。例如，DIM a(m to n)语句是错误的，运行程序时将显示出错信息。

(5) DIM 语句格式中的“〈下界〉to〈上界〉”可以写成“〈下界〉:〈上界〉”的形式。定义数组时，若下界为1，“〈下界〉to”可以省略。

```
DIM   score(1 to 100, 1 to 4)
DIM   score(1:100, 1:4)
DIM   score(100,4)
```

以上三个 DIM 语句是等价的。

(6) 数组定义之后，可以用 MAT 簇重新定义或调整数组某维的上下界，实现数组大小的动态调整。

〔例6-1〕　数组的输入与输出。

```
! Input and ouput of array
DIM array(10)
FOR i=1 to 10
    READ array(i)
```

```
NEXT i
FOR i=1 TO 10
     PRINT array(i);"      ";
NEXT i
PRINT
FOR i=10 TO 1 STEP -1
     PRINT array(i);"      ";
NEXT i
PRINT
FOR i=1 TO 10 STEP 2
     PRINT array(i);"      ";
NEXT i
DATA 20,17,24,50,47,82,35,8,66,39
END
```

程序中共使用了四个 FOR 循环。第一个循环实现自数据区向数组各元素提供数据，第二个循环表明按数据的原有顺序显示，第三个循环则实现按逆序显示，第四个循环通过引用下标为奇数的元素，以显示出数据区中第1,3,5,7,9个数据。

该程序还说明了 True BASIC 引入数组的必要性，以及访问数组元素的方法。可以设想，如果一个程序需要处理的数据量多至上千或者更多，并且仅用简单变量来实现按逆序引用数据区中的数据，其程序代码的长度将会达到人们无法承受的程序。相反，引入数组后，仅需改变循环变量的初值和终值即可。

另外，读者还应注意到循环变量的可能取值，以及循环体内引用数组元素的形式。例如，将第一个循环中的“READ array(i)”改为“READ array(11-i)”，重新运行程序后显示的结果将发生改变。请读者思考会产生什么变化。

〔例6-2〕 按10-19,20-29,30-39,40-49,50-59,60-69,70-79等七个年龄区间，统计某街道办事处所管辖区域10-79岁居民的年龄分布。

分析：一个居民的年龄属于哪个区间可以通过简单计算求得。例如，年龄 age=29，必有 int(age/10)=2，即为第二个区间中的年龄。为统计年龄分布，需定义7个变量以便对每个区间的人数计数。为此，定义一个大小为7的一维数组，每个元素作为一个计数器使用。

算法如图6-3所示。

程序如下：

```
! Count the distribution of ages
DIM distr (7)
FOR i=1 TO 7
     LET distr(i)=0
NEXT i
DO WHILE MORE DATA
    READ age
    LET interval=int(age/10)
```

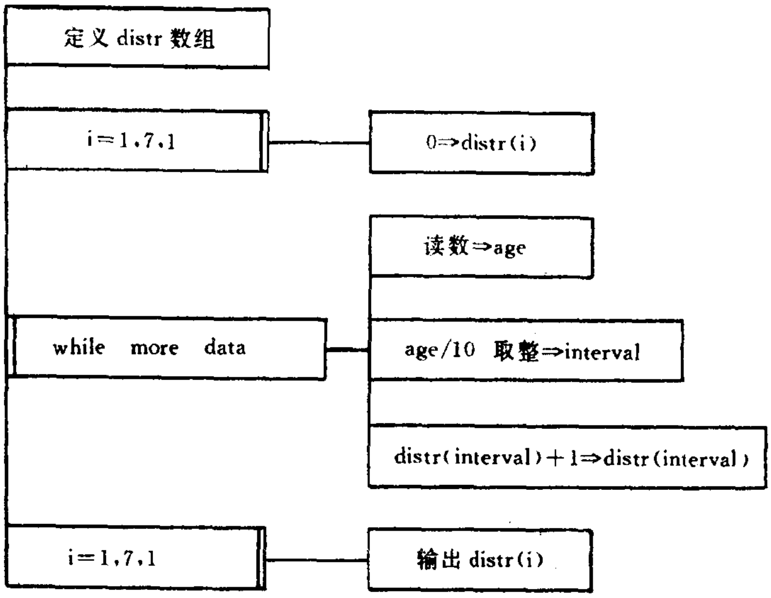

定义 distr 数组
i=1,7,1
0⇒distr(i)
读数⇒age
while more data
age/10 取整⇒interval
distr(interval)+1⇒distr(interval)
i=1,7,1
输出 distr(i)

图6-3 统计居民年龄分布算法

```
    LET distr(interval)=distr(interval)+1
LOOP
PRINT "Distribution of the age"
PRINT "——————————————————"
FOR i=1 to 7
    PRINT 10*i; "———";i*10+9, distr(i)
NEXT i
PRINT "——————————————————"
DATA 25,34,54,61,42,24,13,33,50,26,17,30,14
DATA 19,41,69,27,36,31,52,41,73,43,24,28
END
```

程序中，巧妙地利用了年龄 age 的取值与数组元素的下标之间存在的关系，实现用赋值语句统计相应区间的居民人数，从而取代了多择一选择结构。为了使读者更深入地体会使用数组所具有的优势，请用7个简单变量作为各区间的计数变量，重新编写此程序。

〔**例6-3**〕 将十进制整数转换为二进制数。

分析：其转换方法为：反复除2取余，直到商等于0，最后将各余数按逆序列出，即得到所求的二进制数。但应注意，每次除2所得余数必须存入变量中。

实现上述算法的方法有多种。该例采用字符串型数组的方法，即每次除法运算得到的余数(0或1)作为字符存入数组元素中，最后按逆序输出数组元素。

其算法如图6-4所示。

程序为：

```
! Change from decimal number to binary number
DIM binnum$(64)
```

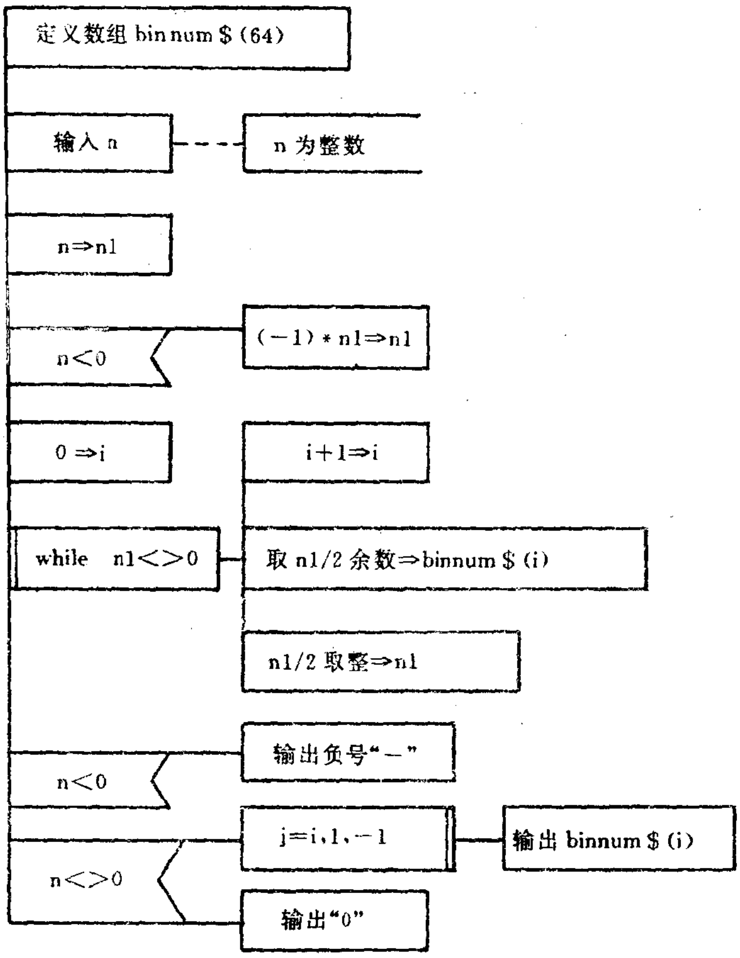

定义数组 binnum $ (64)
输入 n
n 为整数
n⇒n1
n<0
(−1) * n1⇒n1
0 ⇒i
i+1⇒i
while n1<>0
取 n1/2 余数⇒binnum $ (i)
n1/2 取整⇒n1
n<0
输出负号"−"
j=i,1,−1
输出 binnum $ (i)
n<>0
输出"0"

图6-4 十进制整数转换为二进制数

```
INPUT prompt "Please enter an integer:":n
LET n1=n
IF n<0 then LET n1=(−1) * n1
LET i=0
DO while n1<>0
    LET i=i+1
    LET binnum $ (i)=str $ (mod(n1,2))
    LET n1=int(n1/2)
LOOP
PRINT str $ (n);"(10)=";
IF n<0 then PRINT "−";
IF n<>0 then
   FOR j=i to 1 step−1
         PRINT binnum $ (j);
   NEXT j
```

```
ELSE
    PRINT "0";
END IF
PRINT "(2)"
END
```

程序说明：语句“LET binnum $ (i)＝str $ (mod(n1,2))”表示先求出 n1除以2的余数，再转换为字符量，然后传送给字符串型数组元素。

二、MAT REDIM 语句

设计通用程序时，常常需要改变已经定义的数组的大小，为了实现这一功能，True BASIC 2.03版本提供了 MAT REDID 语句。该语句格式为：

```
MAT REDIM 〈数组1〉(〈下界11〉 to 〈上界11〉〔,〈下界12〉 to 〈上界12〉〕…),…
```

其中，上下界可以是常数，变量或数值表达式，其他与 DIM 语句相同。执行该语句时，将根据给定数据量的实际值重新定义数组大小。

〔**例6-4**〕 某系 n 名学生参加了英语四级考试，编程求平均分数，并将超出平均分数的学生及成绩输出显示。

分析：n 名学生的姓名和考试成绩均需要保存，以便与计算出的平均成绩比较，决定是否应输出显示。因此，将所有数据组织成 name $ 和 english 两个一维数组。另外，鉴于编程时无法确定学生的具体人数，因此，必须先对数组作形式定义，然后根据键盘输入的学生人数用 MAT REDIM 语句重新定义数组的大小。算法为：

```
s1：形式定义数组 name $ 和 english。
s2：输入学生人数 n。
s3：重新定义 name $ 和 english 数组。
s4：输入 n 个学生的姓名和英语成绩。
s5：求平均成绩 aver，且输出。
s6：将 n 个成绩逐一与 aver 比较，显示超出平均分的姓名和成绩。
```

程序为：

```
DIM name $ (1),english(1)
INPUT prompt "How many students?":n
MAT redim name $ (n), english(n)
LET sum=0
FOR i=1 to n
     PRINT "Enter name and score:";
     INPUT prompt "———→":name $ (i),english(i)
     LET sum=sum+english(i)
NEXT i
LET aver=round(sum/n,1)
PRINT repeat $ ("   ",2);"Report of higher than average"
PRINT "———————————————————"
PRINT "        name        score"
PRINT "———————————————————"
FOR i=1 to n
    IF english(i)>aver then
       PRINT "        ";name $ (i);"          ";english(i)
       PRINT "———————————————————"
    END IF
NEXT i
END
```

说明:(1) 利用 MAT REDIM 语句重新定义的数组,事先必须经 DIM 语句进行定义,并且不能改变其维数和类型。若不需要改变某维的下界时,"〈下界〉 to"部分可以省略。

(2) 重新定义数组后,并未清除数组各元素原来的值,而是按照行的顺序将原来的值在重新定义后的数组中再次排列。重排时,多出的数据被弃掉,不足时置零(数值型数组)或置空串(字符串型数组)。

三、数组测试函数

程序设计时,既可以利用 MAT REDIM 语句重新定义数组,也可以利用下节介绍的 MAT READ、MAT INPUT、MAT 语句动态调整数组的大小。为实现对数组的有效处理,True BASIC 提供了一组与数组有关的函数。

size(A)	测试数组 A 的元素个数。
size(A,n)	测试数组 A 第 n 维的元素个数。
Lbound(A)	测试一维数组 A 的下界。
Lbound(A,n)	测试数组 A 第 n 维的下界。
Ubound(A)	测试一维数组 A 的上界。
Ubound(A,n)	测试数组 A 第 n 维的上界。

第三节　数组的整体输入与输出

True BASIC 提供的 MAT 簇语句除了可以实现数组重新定义以外,还可以实现数组的"整体性"输入和输出,以及数组赋值和其他运算。

一、MAT READ 语句

语句格式为:

```
MAT READ 〈数组1〉,〈数组2〉,…,〈数组 n〉
```

功能:自 DATA 语句构成的数据区中为一个或多个数组读入所需的数据。例如,执行:

```
DIM a(10),b(3,4)
MAT READ a, b
```

等价于执行下面的程序段：

```
DIM a(10),b(3,4)
FOR i=1 to 10
    READ a(i)
NEXT i
FOR i=1 to 3
    FOR i=1 to 4
        READ b(i,j)
    NEXT j
NEXT i
```

可见，执行 MAT READ a,b 语句时，首先自数据区读10个数据且传送给 a 数组相应的元素中，然后再为 b 数组读所需的12个数据。因此，在 DATA 语句中应首先书写 a 数组所需的10个数据，接着再书写 b 数组所需的数据。

MAT READ 语句的另一个功能是可以动态调整数组大小。实现方法是在语句中的数组名后，使用变量或表达式重新定义数组各维的上下界。

鉴于数组的大小是由变量或表达式的值决定的，所以这种调整又称为固定方式。但读者应当注意：① 调整时不能改变原数组的维数和类型，并且原有的 DIM 语句是不可省略的。② 数据区中必须有相应个数的数据，以满足为数组读取数据的需要。③ MAT READ 与 MAT REDIM 语句之间的区别在于前者需要自数据区为数组所有元素提供数据，而后者由系统对新增的数组元素作初始化处理，原有数组元素的值仍然保留。

〔**例6-5**〕 调整数组大小举例。

```
! Adjust the size of array
DIM a(1,1)
MAT READ a(2 to 3, 5 to 7)
LET l1=lbound(a,1)
LET u1=ubound(a,1)
LET l2=lbound(a,2)
LET u2=ubound(a,2)
PRINT “l1=”;l1,“u1=”;ul
PRINT “l2=”;l2,“u2=”;u2
FOR i=11 to ul
    FOR j=12 to u2
        PRINT a(i,j),
    NEXT j
    PRINT
NEXT i
DATA 6,5,4,3,2,1
END
```

程序中，MAT READ a(2 to 3,5 to 7)语句对数组每一维的上下界重新进行了定义。为输

出经调整后的数组，程序中使用了数组测试函数。

注意，执行具有调整数组大小功能的 MAT READ 时，系统先调整数组大小，然后自数据区读取数据且传送给数组所有元素。

二、MAT PRINT 语句

MAT PRINT 语句用来实现整个数组的输出显示。其一般格式为：

```
    MAT PRINT 〈数组1〉,〈数组2〉,…
 或 MAT PRINT 〈数组1〉;〈数组2〉;…
```

说明：格式中的“,”或“;”，其作用是使标点符号前的数组其各元素按标准格式或紧凑格式输出。若最末一个数组需要按标准格式输出，其后的“,”可以省略；若要按紧凑格式输出，其后必须有分号“;”。

利用一个 MAT PRINT 语句输出多个数组时，每个数组输出之后要自动空一行。每个数组各元素的输出是按在内存中排列的顺序输出的。对二维数组而言，每行元素输出后都自动换行。对三维数组，每输出一个二维也要空一行。

〔**例6-6**〕 MAT PRINT 语句输出格式举例。

```
! Print 1-dim,2-dim and 3-dim array
DIM a(6),b(2,3),c(2,3,3)
MAT READ a,b,c
MAT PRINT a;b;c;
DATA 1,2,3,4,5,6
DATA 1,3,5,7,9,11
DATA 2,4,6,8,10,12,14,16,18,17,15,13,11,9,7,5,3,1
END
```

```
———————————————————————————————— ex06-06 ————————————————————————————————
OK. run
1   2   3   4   5   6
1                        3                        5
7                        9                        11


2   4   6
8   10   12
14   16   18


17   15   13
11   9   7
5   3   1
```

MAT PRINT 语句不具有计算功能，即不能利用此语句先对多个数组进行运算，然后再输出。此外，该语句不具有调整数组大小功能。

三、MAT INPUT 和 MAT LINE INPUT 语句

1. MAT INPUT 语句

该语句用来实现自键盘为数组输入数据。其语句格式为：

```
   MAT INPUT  〈数组1〉,〈数组2〉,…
或 MAT INPUT PROMPT "〈提示字符串〉":〈数组1〉,〈数组2〉,…
```

〔**例6-7**〕 MAT INPUT 语句应用举例。

```
! Calculate the sum of all elements in array
DIM a(2,3)
MAT INPUT prompt "Please enter the 6 values:";a
LET sum=0
FOR i=1 to 2
    FOR j=1 to 3
        LET sum=sum+a(i,j)
    NEXT j
NEXT i
PRINT "sum=";sum
END
```

执行 MAT INPUT 语句与 INPUT 语句的不同之处在于：按行输入数组所需的数据，即一个数组所需的数据应该在一行内输入，在下一行为第二个数组输入数据。若一行内容纳不下需要输入的数据，可在该行最末一项数据后加逗号，并按回车键，表示继续为该数组输入数据。其他情况两语句完全相同，不再赘述。

MAT INPUT 语句具有调整数组大小的功能，并且可以采取两种方式调整数组大小。第一种方式为固定方式，实现方法与 MAT READ 语句相仿。例如：

```
DIM a(5)
INPUT PROMPT "n=":n
MAT INPUT a(n)
```

在 MAT INPUT 语句中对 a 数组重新进行了调整，其大小由运行程序时事先为变量 n 提供的值决定。

第二种调整数组大小的方式为随机方式。具体方法是在数组名后的括号中使用“?”。如：

```
DIM a(5)
MAT INPUT a(?)
```

数组 a 的大小完全由运行程序时，自键盘输入数据的个数确定，这也就是“随机”两字的由来。但读者应注意，以随机方式调整数组大小的功能仅适用于一维数组。

2. MAT LINE INPUT 语句

该语句称为数组整行输入语句，其功能是将自键盘输入的一行字符传送给字符串型数组的一个元素，直到输入足够的行为止。语句格式为：

```
MAT LINE INPUT 〔PROMPT "〈提示字符串〉" :〕〈数组1〉,〈数组2〉,…
```

说明：① 格式中的数组必须是字符串型数组，而不能是数值型数组。

② 自键盘输入数据时，一个输入行内可以包含有空格，逗号和“!”，其中的所有字符均被传送到一个数组元素中。

③ 该语句也具有调整数组大小的功能，但只能采用固定方式，而不能用随机方式。

〔**例6-8**〕 屏幕显示通讯录。

```
! Print the address book
DIM address $ (1)
INPUT prompt "How many items?":n
MAT LINE INPUT prompt "address ——————→": address $ (n)
PRINT "                address book"
PRINT "————————————————————————————"
FOR i=1 TO n
PRINT address $ (i)
PRINT "————————————————————————————"
NEXT i
END
```

第四节 数组赋值与运算

一、数组赋值语句

数值赋值是指把一个特定的值赋给数组各元素，或者把一个数组各元素的值赋给另一个数组的对应元素中。这一操作由 MAT 语句实现，其格式为：

```
   MAT  〈数组〉=〈表达式〉
或 MAT  〈数组1〉=〈数组2〉
```

其中，赋值号“=”左、右两侧的数组分别称为目标数组和源数组。它们应有相同的维数。

例
```
DIM a(5),b$(10)
MAT a=100
MAT b$ =""
```

表明将100赋给数组 a 的各元素，以及使字符串型数组 b$ 的各元素都赋以空串。

使用 MAT 语句可以很方便地实现数组的初始化，利用本节将要介绍的矩阵函数可使数组元素获得预期的初始值。但读者应当注意，赋值号左右两侧的数据类型必须一致。其次，若赋值号右侧表达式中含有运算符，应将整个表达式用圆括号括起来。再者，如果源数组与目标数组大小不同时，系统将自动调整目标数组各维的上界（不改变下界），使之与源数组的大小相同。

〔**例6-9**〕 用 MAT 语句调整目标数组大小示例。

```
! Adjust the size of the array with mat statement
```

```
DIM a(3 to 5, 3 to 5),b(4,6)
MAT b=10
MAT a=b
PRINT “array        lower bound        upper bound        size”
PRINT “——————————————————————————————”
PRINT “        a”;“            ”;lbound(a,1);“              ”;
PRINT ubound(a,1)
PRINT “              ”;lbound(a,2);“              ”;
PRINT ubound(a,2);“              ”;size(a)
PRINT “        b”;“            ”;lbound(b,1);“              ”;
PRINT ubound(b,1)
PRINT “              ”;lbound(b,2);“              ”;
PRINT ubound(b,2);“                ”;size(b)
PRINT “——————————————————————————————”
END
```

————————————————————————— ex06-09 —————————————————————————

```
OK. run
array        lower bound        upper bound        size
—————————————————————————————————
  a            3                   6
               3                   8                24
  b            1                   4
               1                   6                24
——————————————————————————————————
```

二、数值量与数组相乘

根据需要，可以将一个常数或数值表达式与一个数组相乘。其语句格式为：

```
    MAT  〈数组1〉=〈数值表达式〉*〈数组2〉
```

其中，数值表达式必须位于数组2的左侧，若表达式不是常数或变量，则必须用圆括号括起来。

〔**例6-10**〕 数值量与数组相乘示例。

```
! Multiply array by the numerical value
DIM a(2,2),b(2,2)
LET n=4
MAT INPUT prompt “please enter 4 values:”:b
MAT a=(1/n)*b
MAT PRINT a
```

```
END
```

ex06-10

```
OK. run
please enter 4 values:1,4,6,5
 .25            1
 1.5            1.25
```

数值量与数组相乘时，同样要求目标数组与源数组的维数相同，并且系统可以自动调整目标数组的上界，使其大小与源数组相同。

三、数组的加、减运算

利用 MAT 语句可以对两个数组进行相加或相减运算，但参加运算的数组必须维数相同、大小相同，而不要求各维下标的上下界也相同。执行运算时，将两个数组对应元素的值相加或相减，并把和或差传送给目标数组的相应元素。其语句格式为：

```
MAT sum=a+b
MAT dif=a-b
```

一个 MAT 语句只能对两个数组求和或差，同时，系统可对目标数组的大小自动进行调整（只调整各维的上界，而下界不变）。

〔**例6-11**〕 计算两数组的和与差示例。

```
! Calculate the sum and the difference of two 2-dim arrays
DIM a(3,2),b(3,2),sum(3,2),dif(3,2)
MAT READ a,b
MAT sum=a+b
MAT dif=a-b
MAT PRINT a,b,sum,dif
DATA 12,10,8,6,4,2
DATA 1,3,5,7,9,11
END
```

四、两矩阵相乘

按照矩阵运算规则，如果 A 是 L * M 矩阵，B 是 M * N 矩阵，才可以进行矩阵相乘运算，并且 A * B 后所得的结果是一个 L * N 矩阵。

〔**例6-12**〕 计算两矩阵相乘示例。

```
! Multiplication of two arrays
DIM a(4,2),b(2,3),c(4,3)
MAT READ a,b
MAT c=a*b
MAT PRINT c
DATA 1,2,3,4,5,6,7,8
DATA 1,2,3,4,5,6
END
```

五、内部数组与矩阵函数

为方便用户编程，True BASIC 设置了一组内部数组（表6-1）。这些数组是用户使用一个数组之前，对数组进行初始化时常用的初始值。例如，全部元素值为零的数组，全部是空串的数组等。在 MAT 赋值语句赋值号的右侧使用内部数组，可以对目标数组赋予某种形式的初值，同时还可以根据需要重新对数组的大小进行调整。

内部数组具有带下标界限和不带下标界限两种格式。利用 MAT 语句向目标数组赋值时，若源数组是带下标界限的内部数组，目标数组的大小将被调整，改变到与内部数组的大小相一致；若使用不带下标界限的内部数组，则赋值后目标数组将维持原来的大小。例如，执行

```
DIM f(1,1)
MAT f=Zer(3,3)
```

目标数组 f 的大小将被调整成3×3的数组。又如，

```
DIM c(5,5)
MAT c=Zer
```

表示仅将 c 数组各元素“置零”，它的大小并未改变。

表6-1　　True BASIC 的内部数组

内部数组	意　　义
con	一个所有元素皆为1的数组，数组的维数和大小与目标数组相同。
con(x,y,…)	一个所有元素皆为1的数组，它是一个 x * y * …数组。
idn	单位矩阵（主对角线元素为1，其余元素为0的方阵），其维数和大小与目标数组相同。
idn(x)	一个 x * x 的单位矩阵。
nul $	一个所有元素均为空串的字符串数组，与目标数组具有相同的维数和大小。
nul $ (x,y,…)	每个元素均为空串的字符串数组，它是一个 x * y * …数组。
zer	每个元素均为0的数组，与目标数组具有相同的维数和大小。
zer(x,y,…)	每个元素均为0的数组，它是一个 x * y * …数组。

除数组测试函数之外，True BASIC 还提供有如表6-2所示的四个与矩阵运算有关的函数。

表6-2　　True BASIC 矩阵函数

函数名	意　　义
trn(x)	x 必须是一个二维数组，函数返回 x 的转置矩阵。即 m×n 的矩阵 x 的转置是一个 n ×m 的矩阵 y，且满足 y(i,j)=x(j,i)。
inv(x)	函数返回方阵的逆矩阵，如果 x 是一个奇异矩阵，或近似于一个奇异矩阵，True BASIC 给出错误信息。
det(x)	函数返回方阵 x 的行列式的值，如果 x 是奇异的，返回值为零。
dot(x,y)	函数返回两个向量 x 和 y 的点积。x 和 y 必须具有相同的元素个数。

注意，trn 和 inv 函数应置于 MAT 语句合适的位置上，而 det 和 dot 函数可根据需要作为数值表达式中的运算分量。

第五节 排 序

排序又称分类。它是程序设计中的一类重要运算，其功能是将一组数据元素的无序序列调整成为一个有序序列。

实际工作中，我们常常要将数据进行比较、排序，使数据达到应用要求。例如短跑运动员的名次是根据运动成绩排序的结果。排序通常是按数据的大小，依照递增或递减的次序进行排序。排序的目的是便于对已经排序的数据进行检索。

由于待排序的数据量不同，可将排序方法分为两大类：一类是内部排序，指的是待排序数据存放在内存中进行的排序过程；另一类是外部排序，指的是待排序数据的数量很大，内存中一次无法容纳全部数据，在排序过程中尚需对外存进行访问的排序过程。本节仅讨论内部排序的程序设计。

实现内部排序的算法很多。但就其全面性能而言，很难提出一种公认是最好的算法。每一种算法都各有特色，适合在不同的环境下使用。分析排序算法效率的指标有三个：一是执行算法的时间开销；二是执行算法所需要的存储空间；三是算法本身的复杂程度。

下面介绍两种最简单的排序算法：选择排序和交换排序。为叙述方便起见，首先假设：①待排序的数据有 n 个，且已传送到一维数组 a 中。②作降序运算。

一、选择排序

算法设计：

s1：将 a(1)和数组元素 a(2)～a(n)依次一一进行比较。若 a(1)<a(2)，就交换这两个变量的值；若 a(1)≥a(2)，则两个变量的值不交换。然后再将 a(1)和 a(3)比较，处理方法类同。依此类推，一直比到 a(n)，这时 a(1)的值就是 a(1)～a(n)的最大值。

s2：找次大值。即将 a(2)和 a(3)～a(n)依次一一比较，比较结束 a(2)就是次大值。

⋮

s_{n-1}：找次大值。将 a(n－1)与 a(n)进行比较，处理方法同上。比较结束使 a(n－1)为次大值，此时，a(n)是 a 数组中的最小值。

为实现上述算法，须设置双重循环。外循环实现共计 n－1轮的比较过程，设循环变量为 i，显然，i 应从1到 n－1依次取值；内循环控制每一轮的比较过程，使 a(i)成为 a(i)，a(i+1)，…，a(n)中的最大值，若循环变量为 j，按照以上算法，j 应从 i+1到 n 依次取值。选择排序算法如图6-5所示。

程序如下：

```
! Selection sort
DIM a(10)
MAT INPUT prompt "Please enter your data items:": a(?)
LET n=ubound(a)
FOR i=1 to n-1
```

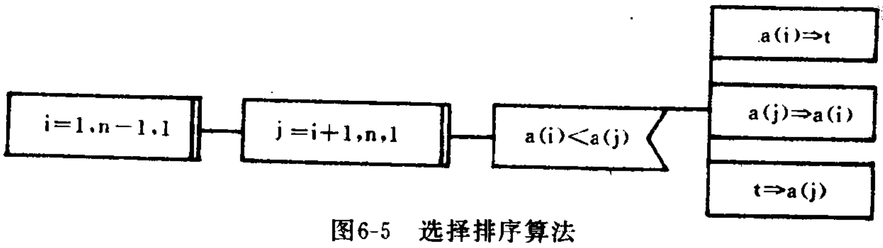


图6-5　选择排序算法

```
          FOR j=i+1 to n
               IF a(i)<a(j) then
                  LET t=a(i)
                  LET a(i)=a(j)
                  LET a(j)=t
               END IF
          MEXT j
     NEXT i
     MAT PRINT a;
     END
```

选择排序的特点是算法简洁，易于实现，其基本操作是比较两个数组元素的大小和交换位置，并且数据比较次数与数据的原有顺序无关。排序过程中，第一轮要比较 n−1次，第二轮要比较 n−2次，……，第 n−1轮要比较1次。总的比较次数为：

$$(n-1)+(n-2)+\cdots+1=n(n-1)/2$$

由于实现两个变量值交换的操作处于内循环的循环体内，因此，用上述算法排序时数据交换次数较多。可以考虑将上述算法稍加修改，方法是：设置一个指针 pointer，在第 i 轮比较过程中，首先把它指向 a(i)，也就是将 a(i)的下标 i 传送给 pointer，然后让该元素与其后各元素逐项比较，在比较过程中总使 pointer 指向较大的元素。这一轮比较结束后，pointer 中的值便是未排序各元素中的最大元素的下标值。这时，若 i 与 pointer 的值相等，说明 a(i)的值即是这轮比较过程中的最大值；若不相等，则需将它们所指向的元素交换，即作 a(i)与 a(pointer)交换。这种经修改后的排序方法就是人们常用的直接选择排序算法。其算法如图6-6所示。

程序为：

```
! Sort by straight selection
DIM a(10)
MAT INPUT prompt "Please enter your data items:";a(?)
LET n=ubound(a)
     FOR i=1 to n-1
          LET pointer=i
          FOR j=i+1 to n
               IF a(pointer)<a(j) then LET pointer=j
         NEXT j
```

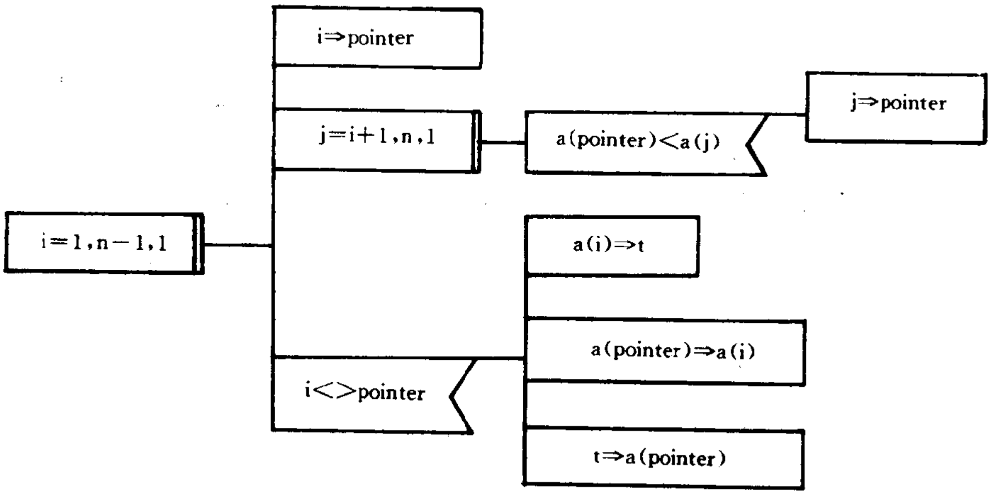

i=1,n-1,1
i⇒pointer
j=i+1,n,1
a(pointer)<a(j)
j⇒pointer
i<>pointer
a(i)⇒t
a(pointer)⇒a(i)
t⇒a(pointer)

图6-6　直接选择排序算法

```
      IF i<>pointer then
         LET t=a(i)
         LET a(i)=a(pointer)
         LET a(pointer)=t
      END IF
   NEXT i
   MAT PRINT a;
   END
```

直接选择排序算法其数据比较次数仍为 n(n－1)/2，但数据交换次数大为减少。如果数据的初始次序为降序时，交换次数为0；为升序时，交换次数为 n－1。

二、交换排序

交换排序的基本方法是：两两比较待排序数列中的数据，并交换不满足顺序要求的那些数据对，直到全部序列排好序为止。常用的交换排序方法是冒泡排序。

冒泡排序的算法是：先比较 a(1)和 a(2)，如果 a(1)<a(2)，则交换它们的值，然后对 a(2)(可能是刚交换来的)和 a(3)作同样的处理，重复此过程直到处理完 a(n－1)和 a(n)。这种从(a(1),a(2))到(a(n－1),a(n))的 n－1次比较和交换的过程称为一次冒泡，其效果是将最小的元素好似水中的汽泡一样，飘浮到水面(即传到了最后位置)，而使其他元素作适当地移动。重复冒泡过程，不过第二次只须从(a(1),a(2))处理到(a(n－2),a(n－1))，执行 n－2次比较即可。这样总共做 n－1次冒泡即可完成数据的排序运算。

具体实现时，可以设置一个标志变量 flag，表示本次冒泡是否出现数据交换。如果没有作交换，则表示在这次冒泡之前数据序列已经达到排序要求，故可以停止处理。

冒泡排序算法如图6-7所示。

程序如下：

```
! Bubble sort
```

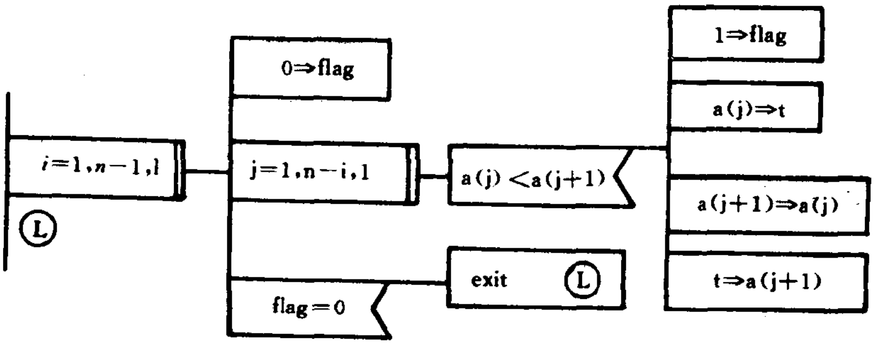


图6-7　冒泡排序算法

```
DIM a(10)
MAT INPUT prompt "Please enter your data items:";a(?)
LET n=ubound(a)
FOR i=1 to n-1
    LET flag=0
    FOR j=1 to n-i
        IF a(j)<a(j+1) then
           LET flag=1
           LET t=a(j)
           LET a(j)=a(j+1)
           LET a(j+1)=t
        END IF
    NEXT j
    IF flag=0 then EXIT FOR
NEXT i
MAT PRINT a;
END
```

分析：如果数列的初始顺序已经按降序排列，则经一次冒泡就可结束排序过程，数据比较次数为 n−1，数据移动次数为零。若数列初始顺序为升序时，比较次数达到最大，为

$$\sum_{i=1}^{n-1} i = \frac{n}{2}(n-1)$$

并且每比较一次都要做一次交换，移动次数也达到最大，为

$$3 \cdot \sum_{i=1}^{n-1} i = \frac{3}{2}n(n-1)$$

这一节共列举出三个排序程序，如果将程序中两个数组元素之间的比较由“<”改为“>”，那么程序便可对数据按相反的顺序(即从小到大，升序)进行排序。其次，如果将程序中的数值型数组改为字符串型数组，程序即可对字符串型数据按 ASCII 码值的大小进行排序运算。

第六节　应用程序举例

一、数组的计算与统计

〔**例6-13**〕　试编程计算下列矩阵用下划线标识的诸元素之和。

$$A=\begin{bmatrix} 18 & 14 & 5 & 27 & 4 \\ \underline{22} & 25 & 10 & 6 & 9 \\ \underline{15} & \underline{23} & 17 & 16 & 24 \\ \underline{19} & \underline{43} & \underline{11} & 7 & 10 \\ \underline{16} & \underline{23} & \underline{41} & \underline{33} & 20 \end{bmatrix}$$

分析：涉及矩阵的计算问题，常常需要分析被处理元素的两个下标之间的关系，以及每个下标的取值范围。本题中求和的诸元素应满足条件：元素所在的行数必须大于列数，行数的取值范围为2到5，列数为1到4。

设变量 i 和 j 分别代表矩阵元素的行下标和列下标。程序可设计为：

```
! Calculate the sum of the partial elements in array
DIM a(5,5)
INPUT prompt “Enter n:”;n
MAT INPUT prompt “Enter the elements of a maxtrix:”:a(n,n)
LET sum=0
FOR i=2 to n
    FOR j=1 to i-1
        LET sum=sum+a(i,j)
    NEXT j
NEXT i
PRINT “sum=”;sum
END
```

为了使程序具有通用性，程序中利用 MAT INPUT 语句对 a 数组的大小进行了调整。但应注意，执行该语句时必须输入数组的所有数据，而不能仅输入一部分数据。

〔**例6-14**〕　设某交电公司所辖四个商店年终销售三种牌号电视机的数量如表6-3所示，每种电视机的单价如表6-4所示。要求计算出各商店的营业额，以及四个商店每种电视机的总销量。

表6-3　　电视机销售量

牌号 销量 商店	北京牌	黄河牌	熊猫牌
商店1	70	80	40
商店2	42	50	32
商店3	65	74	52
商店4	100	96	85

表6-4　　单价表

价格 牌号	元
北京牌	2450
黄河牌	2500
熊猫牌	2170

分析：计算营业额属于两矩阵相乘问题，为此，定义四个数组：sale(4,3)存放各商店电视机销售数据，数组 price(3)存放电视机单价，数组 amt(4)存放各商店营业额，数组 total(3)存放每种电视机总销量。

程序为：

```
! Calculate sales amount of every shop
! and total sales volume
DIM sale(4,3),price(3),amt(4),total(4)
MAT READ sale, price
MAT amt=sale * price
PRINT "           shop sales amount"
PRINT "-----------------"
MAT PRINT amt;
MAT total = zer
FOR i=1 to 3
    FOR j=1 to 4
        LET total(i)=total(i)+sale(j,i)
    NEXT j
NEXT i
PRINT "total sales volume"
PRINT "------------"
MAT PRINT total;
DATA 70,80,40,42,50,32
DATA 65,74,52,100,96,85
DATA 2450,2500,2170
END
```

二、查 找

为了便于讨论，首先引入“关键字”的概念。关键字是数据元素中某个数据项的值，用它可以标识一个数据元素。若此关键字可以唯一地标识一个数据元素，则称此关键字为主关键字。反之，称用以标识若干数据元素的关键字为次关键字。

需要说明的是，一个数据元素可由若干个数据项组成。例如，n 名学生参加数学、物理、化学三门课程考试，若将学号及各科成绩组织成 n×4的二维数组，数组的每一行则为一个数据元素，表明是一个学生的各项数据。按照关键字的概念，学号为主关键字，各科成绩均为次关键字。然而，一个数据元素也可以只含一个数据项，此时关键字即为数据元素的值。

查找又称为检索，是指根据给定的某个值，在一个含有众多的数据元素的查找表中确定一个其关键字等于给定值的数据元素。若表中存在这样的数据元素，则称查找是成功的，此时查找的结果为显示或打印整个数据元素的信息，或给出该数据元素在查找表中的位置；若表中不存在关键字等于给定值的数据元素，则称查找不成功，此时查找的结果可给出“找不到”的信息。

查找的方法随数据结构不同而不同。若数据组织成数组，对无序数组和有序数组可采用不同的查找方法，其查找效率有明显不同。第九章将讨论基于数据文件的查找方法。下面仅就数组的顺序查找和二分法查找进行讨论。

1. 顺序查找

对无序数组而言，查找时只能采用顺序查找方式。其查找过程为：从数组的第1个数据元素开始，逐个进行数据元素的关键字与给定值的比较，若某个数据元素的关键字和给定值相等，则查找成功，找到所查数据元素；反之，若遍历整个数组，其关键字和给定值比较均不相等，则表明查找失败。

需要补充说明的是，对上述查找过程，不论查找成功还是查找失败，均要输出相应信息，或者是查找到的数据元素的值，或者是“未找到”信息。其次，对主关键字查找时，若查找功能，输出查到的数据元素的值后，即可结束查找过程；对次关键字查找时，若查找成功，尚需继续查找。

〔**例6-15**〕 以产品号作为主关键字，顺序查找库存物品。

设仓库有 n 种物品，每种物品包括：产品号、库存量、单价等项数据，全部数据组织成二维数组。其中产品号为第1列数据。

程序为：

```
! Sequential search
DIM stock(1,3)
INPUT prompt "How many kinds goods?":n
MAT input stock(n,3)
INPUT prompt "Enter the product number to search:":prod_num
LET flag=0
FOR i=1 to n
    IF stock(i,1)=prod_num then
       LET pointer=i
       LET flag=1
       EXIT FOR
    END IF
NEXT i
IF flag=1 then
   PRINT "-------------------------"
   PRINT "product-No        reorder quan        unit price"
   PRINT "-------------------------"
   PRINT stock(pointer,1),stock(pointer,2),stock(pointer,3)
ELSE
    PRINT prod_num; "has not found!"
END IF
END
```

程序中，利用 FOR 循环结构实现顺序查找。在查找过程中，若给定产品号与某一物品的产

品号相等，记下该物品的下标值(stock 的第1个下标，即 i 的值)，并将查找成功标志 flag 置1(其初始值为0)，然后结束循环。若遍历整个数组仍未找到，标志变量 flag 仍维持初始值。因此，在 FOR 循环结构之后，可根据变量 flag 的不同取值输出相应的信息。

顺序查找算法的实质是枚举。这一方法仅适用于数据量不大的情况，其平均查找次数为 n/2。显然，数据量越大，查找的平均次数就越多。

2. 对分查找

对分查找亦称二分法查询，是一种基于有序数组的查找方法。其查找过程是：先确定待查数据元素所在的范围(区间)，然后逐步缩小区间，直到查找成功或找不到该数据元素为止。

假设数组按降序排列，并且指针 low 和 high 分别指示待查数据所在范围的下界和上界，low 和 high 的初始值分别为1和 n(数组第1个元素和最后一个元素的下标值)，指针 mid 指示区间的中间位置，即 mid＝int((low＋high)/2)。

查找算法：

S1：将查找范围中间位置的数据元素的关键字与给定值比较，如果前者值大，说明给定值若存在于数组中，必在区间〔mid＋1，high〕范围内，执行 mid＋1⇒low；若前者值小，说明给定值如果存在，必在区间〔low，mid－1〕范围内，执行 mid－1⇒high；若两者相等，则查找成功，所查元素在数组中的位置等于指针 mid 的值。

S2：在新的查找范围内重复上面的处理过程，即重新计算查找范围的中间位置(求 mid 的值)，并按 S1类似的方法进行比较，决定查找成功还是把查找范围缩小一半，继续查找，……。继续查找的条件是查找不成功并且 low≤high。如果 low 大于 high，说明数组中不存在有数据元素的关键字与给定值相等，即查找不成功。

对一维数组 a 实施对分查找的算法描述如图6-8所示。

〔**例6-16**〕 秀峰厂对所属各分厂进行安全生产评比，统计数据如下：

分厂编号	100	107	123	420	…	158
安全生产天数	374	224	546	394	…	450

编写程序实现基于分厂编号的对分查找。

程序为：

```
! Binary search
DIM safe_p(10,2)
INPUT prompt "Enter the number of factories:";n
MAT READ safe_p(n,2)
 ! Straight selection sort
FOR i=1 to n-1
    LET pointer=i
    FOR j=i+1 to n
        IF safe_p(pointer,1)<safe_p(j,1) then LET pointer=j
    NEXT j
    IF i<>pointer then
```

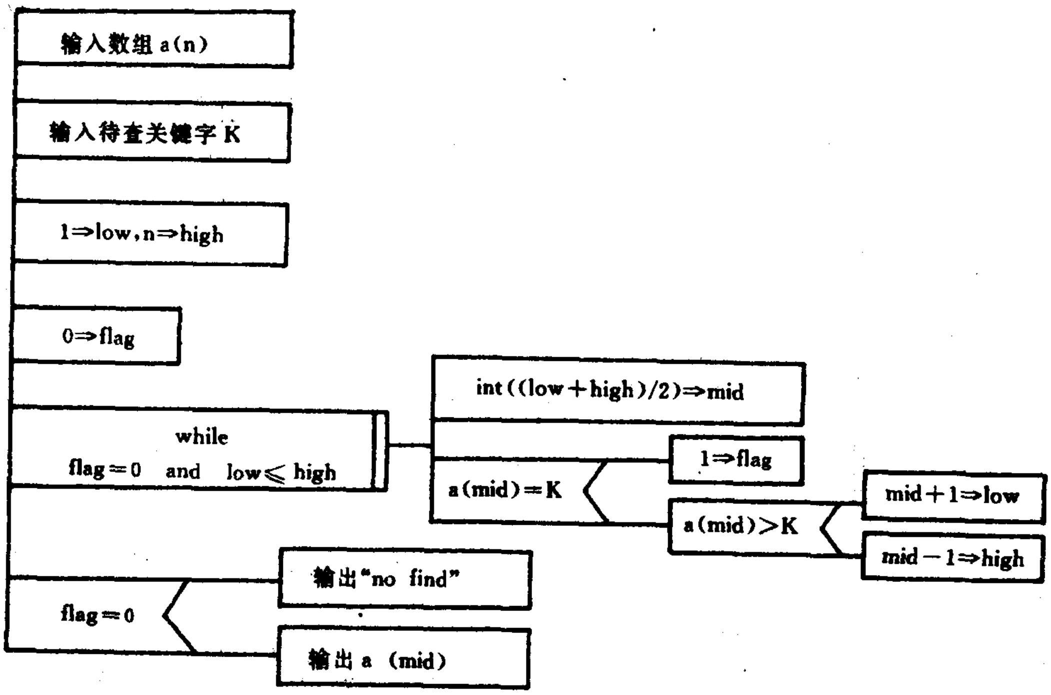

输入数组 a(n)
输入待查关键字 K
1⇒low,n⇒high
0⇒flag
while
flag=0 and low≤high
int((low+high)/2)⇒mid
a(mid)=K
1⇒flag
a(mid)>K
mid+1⇒low
mid−1⇒high
flag=0
输出"no find"
输出 a (mid)

图6-8 基于数组的对分查找算法

```
        LET t1=safe_p(i,1)
        LET safe_p(i,1)=safe_p(pointer,1)
        LET safe_p(pointer,1)=t1
        LET t2=safe_p(i,2)
        LET safe_(i,2)=safe_p(pointer,2)
        LET safe_p(pointer,2)=t2
    END IF
NEXT i
! Look up
INPUT prompt "Enter the branch factory-no to find:":bf_no
LET low=1
LET high=n
LET flag=0
DO
    LET mid=int((low+high)/2)
    IF safe_p(mid,1)=bf_no then
        LET flag=1
    ELSE if safe_p(mid,1)>bf_no then
        LET low=mid+1
    ELSE
        LET high=mid-1
        END IF
    LOOP while flag=0 and low<=high
    IF flag=0 then
       PRINT bf_no;"be not found!"
    ELSE
       PRINT bf_no;"the days of safely is";safe_P(mid,2)
    END IF
    DATA 100,370,107,108,132,645,411,394,158,562
    END
```

与顺序查找相比，对分查找要有效得多，其平均查找次数为 int(log2(n)+1)。如果 n=1000，则平均查找次数为10，这远远小于顺序查找的平均次数500次。

三、数组的插入与删除

所谓数组的插入是指在有序数组中插入给定的数据元素，并使数组的有序性保持不变。算法设计的关键在于：利用 MAT REDIM 语句重新定义有序数组，根据欲插入元素的值查找应插入的位置，然后将该位置以后各元素向后移一位，最后将欲插入元素的值传送到该位置上。

与数组的插入相反，数组的删除是指从有序数组中删除指定的数据元素。其实现方法是：根据给定值查找要删除数据元素的位置，然后将该位置以后各元素向前移一位，最后利用 MAT REDIM 语句重新定义该有序数组。

下面举例说明数组的插入运算。

〔**例6-17**〕 表6-5是按总分由大到小排列的学生成绩有序表。

表6-5 **学生成绩有序表**

学号	数学	物理	化学	总分
736122	80	70	100	250
736113	85	75	80	240
736103	65	75	80	220
⋮	⋮	⋮	⋮	⋮
736133	70	55	75	200

现有一个学生的数据为：736140,81,78,83,242。要求编写程序将该学生数据插入有序表中，并保持成绩表原有排列次序。

算法分析：将成绩有序表组织成 n 行五列的二维数组 score。为实现插入运算，需要把此数组重新定义为 n+1行五列。确定被插入行应插入到 score 数组的位置时，首先置指针 pointer 初始值为 n+1，然后将 total(被插入行的总分)与成绩表各行总分依次比较，一旦确定插入位置，就把当前行的值传送给 pointer，若比较过程中未找到应插入的位置，说明该行应插入到数组的第 n+1行。因此，查找过程结束后判断 pointer 是否不等于 n+1。若不相等，表明该行应插入到数组第 n+1行之前，此时，把第 n 行至 pointer 所指行向第 n+1行至 pointer+1行逐行传送。最后将该行传送到 pointer 所指行。若 pointer 的值等于 n+1，直接将该行数据进行传送即

可。

算法可设计为：

s1：形式定义数组 score(1,5)。

s2：输入成绩表的实际行数 n。

s3：调整 score 为 n×5数组，且自键盘为数组输入数据。

s4：输入欲插入行各项数据。

s5：将 score 重新定义为 n+1行5列数组。

s6：指针 pointer 置初始值 n+1。

s7：利用顺序查找算法确定插入 score 中的位置。

s8：如果 pointer 不等于 n+1，将自第 n 行至第 pointer 行向第 n+1行至第 pointer+1行作对应行传送。

s9：将插入行传送到数组 score 的第 pointer 行。

s10：输出显示 score 数组各元素的值。

程序为：

```
! Insert a new line into table sorted order
DIM score(1,5)
INPUT prompt "Enter the number of line in score table:":n
MAT INPUT prompt "Enter the table sorted order:":score(n,5)
PRINT "Enter the datas of a new student:"
INPUT st_no,math,phys,chem,total
MAT REDIM score(n+1,5)
! Finding location to insert
LET pointer=n+1
FOR i=1 to n
      IF score(i,5)<total then
         LET pointer=i
         EXIT FOR
      END IF
NEXT i
IF pointer<>n+1 then
   FOR k=n to pointer step -1
         FOR l=1 to 5
               LET score(k+1,l)=score(k,l)
         NEXT l
   NEXT k
END IF
! Insert a new line
LET score(pointer,1)=st_no
LET score(pointer,2)=math
LET score(pointer,3)=phys
LET score(pointer,4)=chem
LET score(pointer,5)=total
! Output score report
PRINT "st_no","math","phys","chem","total"
PRINT repeat $ ("-",70)
MAT PRINT score
PRINT repeat $ ("-",70)
PRINT
END
```

习　题

6—1. 数组 w 按如下形式存放各数据：

$$w = \begin{bmatrix} 17.56 & 9.12 & 125.03 & 0.15 & 77 \\ 20.1 & 14.9 & 13.12 & 40 & 1.6 \\ 71.8 & 5.2 & 99.4 & 8.14 & 20.9 \end{bmatrix}$$

若变量 i,j,k 的值分别为1,2,2，试写出下列数组元素所代表的值。

w(i,j)　　w(i+j,k)　　w(k-1,4)

w(i,i+j+k)　　w(k,2)　　w(i,j+k)

6—2. 问答：

(1) 将程序处理的对象组织成数组有什么特点？

(2) 语句 DIM a(101 to 140),b(5 to 8,-2 to 2)定义的数组包含了哪些数组元素？

(3) DIM 语句与 MAT REDIM 语句之间有何区别?MAT REDIM 与 MAT READ、MAT INPUT 语句的异同是什么?MAT 语句与 MAT READ、MAT INPUT 语句的区别又是什么？

(4) MAT PRINT 语句与 PRINT 语句有何区别？

6—3. 写出下列程序的运行结果。

(1)
```
DIM a(6)
FOR i=1 to 6
    LET a(i)=i+10
    PRINT a(i),
NEXT i
END
```

(2)
```
DIM a(2,3)
FOR i=1 to 3
    FOR j=1 to 3
        LET a(i,j)=i+j
        PRINT a(i,j),
          NEXT j
          PRINT
     NEXT i
     END
(3)  DIM a(15),b(3,5)
     FOR i=1 to 15
          READ a(i)
     NEXT i
     FOR i=1 to 3
          FOR j=1 to 5
               LET k=4*(i-1)+j
               LET b(i,j)=a(k)
               PRINT b(i,j);“      ”;
          NEXT j
          PRINT
     NEXT i
     DATA 1,2,3,4,5,6,7,8,9,0,-1,-2,-3,-4,-5
     END
```

6—4. 已知数组

$$A=\begin{bmatrix}3 & 12 & 60\\ 18 & 12 & 26\\ 71 & 65 & 100\end{bmatrix}\qquad B=\begin{bmatrix}104 & 30 & 26\\ 44 & 28 & 73\\ 90 & 46 & 22\end{bmatrix}$$

试编制程序求数组 A 与数组 B 的和 C，并求数组 C 中的最大值，以及最大值的位置。

6—5. 设二维数组 a(n,n)已被赋值，试编写程序片段，计算：

(1) 所有元素之和。

(2) 数组的靠边元素之和。

(3) 两条对角线各元素之和与之积。

6—6. 某门课程考试采用5分制记分。试统计全系 n 名学生中得5分，4分，3分，2分，1分，0分者各有多少人，并计算他们分别占所有人数的百分比。

6—7. 某学习小组共有9名同学，学号分别为1,2,3,…,9，期末考试后各科及格者(学号)如下：

数学：3,1,7,9,6,4,8。

物理：2,5,9,1,7,3,4。

化学：6,7,3,5,4,9,1,8。

英语：2,6,9,4,3,7,8,5,1。

试统计四门课程全部及格者，输出他们的学号。

6—8. 表6-6为五位工人生产四种零件的数量，试编程计算五位工人生产的每种零件的总量，以及每位工人生产四种零件的总和。

表6-6　　　　　　　　　　　　　**生产统计表**

零件 数量 工人编号	零件1	零件2	零件3	零件4
2101	12	22	20	16
2102	8	24	26	18
2103	11	19	22	18
2104	9	21	20	19
2105	10	20	18	18

6—9. 写出下列程序的运行结果。

```
(1) DIM a(5,5)
    MAT a=idn(5)
    MAT a=3*a
    MAT PRINT a
    END
(2) DIM a(100)
    MAT READ a(10)
    FOR i=1 to 10
        LET a(i)=i+a(i)
    NEXT i
    MAT a=a+a
    MAT PRINT a
    DATA 10,9,8,7,6,5,4,3,2,1
    END
(3) DIM a(3,4),b(4,3)
    MAT READ a
    MAT b=trn(a)
    MAT PRINT b
    DATA 10,12,14,16,18,20
    DATA 22,24,26,28,30,32
    END
```

6—10. 用 MAT 簇语句求解方程组：

$$\begin{cases} x_1 + x_2 + 2x_3 + 3x_4 = 1 \\ x_1 + 2x_2 + 3x_3 - x_4 = -4 \\ 3x_1 - x_2 - x_3 - 2x_4 = -4 \\ 2x_1 + 3x_2 - x_3 - x_4 = 6 \end{cases}$$

提示：方程组可写成矩阵形式 AX=B，A 是系数矩阵，X 是未知列向量，B 是常数列向量，则有 $X=A^{-1}B$。其中 A^{-1}是矩阵 A 的逆阵。

6—11. 某次足球邀请赛，有 CHINA（东道主），IRAN，KUWAIT，NEWZEALAND， THAILAND，INDIA 六国参加，按照惯例，东道主在入场式上排在最后，其他各队按英文字母的顺序排列先后。编程完成这项工作。

6—12. 虹达商场复印机部11月份销售复印机情况如下：

品　名	零售价(元)	销售量(台)
理光 FT-5560	68000	15
理光 FT-4088	32800	9
施乐1027	29860	11
佳能 NP-5511	49000	6
东芝 BD-5511	24800	14
优美3200	31900	19

编程统计该月销售总额及上交税额(税额＝销售额的20%)，并按照销售额从高到低输出复印机销售情况表。

6—13. 设某专业 n 名学生参加数学、外语竞赛，每个学生的数据包括：学号，姓名，数学和外语成绩。试编制程序实现如下功能：

(1) 分别计算全部参赛学生数学和外语的平均成绩。

(2) 计算每人的平均参赛成绩，并按平均成绩由高到低排名次。

(3) 任给一个分数线，显示出平均参赛成绩超出此分数线的学生。

6—14. 共有16人参加校田径运动会100米短跑。编制程序实现：

(1) 按成绩排名次(升序)。

(2) 输出显示取得前三名的运动员号码，姓名和成绩。

(3) 基于运动员号码的对分检索。

6—15. 任给一个数据，若它存在于一维数组(按升序排列)中，则将它删除；若找不到给定数据，则显示“没有此数据，无法删除”。试编程实现之。

6—16. 男子乒乓球团体赛规定双方各出三名运动员，以九盘五胜制决定胜负，全部打满共有九盘，若一方先胜五盘即可结束比赛。双方运动员出场顺序见表6-7和表6-8。在中国队与瑞典队的一次比赛中，中国教练根据中国出场运动员对瑞典出场运动员的胜率(见表6-9)和瑞典运动员排定的位置(瑞典队为 B 方，3号打第 Ⅰ 位置，1号打第 Ⅱ 位置，2号打第 Ⅲ 位置)，欲决定中国队各出场运动员的排位，请设计程序输出中国队在所有的排位情况下与瑞典队比赛的结果，供教练参考。

表6-7　A 方出场顺序

位置	场　次		
Ⅰ	2	4	7
Ⅱ	3	6	8
Ⅲ	1	5	9

表6-8　B 方出场顺序

位置	场　次		
Ⅰ	3	5	7
Ⅱ	1	4	8
Ⅲ	2	6	9

表6-9　胜率表

		瑞　典		
		1	2	3
中国	1	0.3	0.5	0.4
	2	0.7	0.65	0.5
	3	0.6	0.45	0.25

第七章　自定义函数、子程序、库和模块

实践证明，求解一个规模较大、复杂程度较高的问题，最有效的方法是按功能将问题分解为若干个子问题，每个程序设计人员仅完成其中的一个或者几个子问题，最后再拼装成一个整体。

实现一个子问题的独立的程序段称为程序的逻辑模块，或称程序部件。运用模块构造程序，是结构化程序设计的重要内容。它可以实现程序的积木化集成，使程序的编写、调试、测试验证在较小的规模上进行，降低程序设计的复杂程度，提高编程效率。

本章将着重介绍构造程序逻辑模块的两种主要结构：由用户定义的函数过程和子程序，并通过实例讲述程序的模块化设计思想。此外，还将说明 True BASIC 程序链接技术，库的建立和调用，以及 True BASIC 2.03版提供的模块结构。

第一节　自定义函数

True BASIC 提供的标准函数，其实质是在系统内部定义有实现特定运算的程序段，用户编程时可直接引用这些函数，以提高编程效率。但是，标准函数的数目是有限的，并且也仅能解决常见的、简单的函数运算，而用户编程时往往会遇到运算关系更为复杂、无法用标准函数描述的函数运算。为此，True BASIC 允许用户根据需要自己定义函数，以供程序中调用。

自定义函数在程序中起着逻辑模块的作用。按照函数定义所处的位置，自定义函数分内部自定义函数和外部自定义函数两种。

为了叙述方便，以下将内部自定义函数和外部自定义函数简称为内部函数和外部函数。

一、内部函数

在 True BASIC 中，主程序、外部函数、外部子程序以及外部图画都被称为程序单位。所谓主程序是指包括有 END 语句的程序，它可以单独运行。其他程序单位不能含有 END 语句，它们不能单独运行，只能通过主程序或调用程序的调用才能被执行。

所谓内部函数是指用户在某一程序单位内部定义的函数，它是该程序单位的组成部分。内部函数有两种定义格式：单行函数和多行函数。

1. 单行函数

用一个 DEF 语句定义的函数称为单行函数。其语句格式为：

```
DEF 〈函数名〉(〈参数1〉[,〈参数2〉]…)=〈表达式〉
```

其中，DEF 是自定义函数语句的关键字，可以写成 FUNCTION。〈函数名〉是函数的名称，其命名规则与简单变量的命名规则相同。自定义函数也有两种类型：数值型和字符串型。若函数值是数值，此函数就是数值型函数；若函数值是字符串，此函数就是字符串型函数。对于字符串型函数，函数名后必须紧跟类型说明符“＄”。〈参数1〉[,〈参数2〉]…是函数的各个自变量，它

们是形式参数，在形态上可以是变量名或数组名。定义函数时，根据需要形式参数也可以没有，即可以定义无参函数。表达式是用户定义的函数运算关系，表达式的值即为函数的值，因此，其类型必须与函数名的类型一致。例如：

```
DEF rndnum=int(100 * rnd+1)
DEF sinh(x)=0.5 * (exp(x)-exp((-1) * x))
DEF hypotenuse(a,b)=sqr(a^2+b^2)
DEF len_shortstring(x$,y$)=min(len(x$),len(y$))
```

以上定义了四个单行函数。其中 rndnum 是无参函数，该函数可产生1～100之间的一个随机整数；函数 sinh 有一个参数，它定义了求自变量 x 的双曲正弦函数的运算；函数 hypotenuse 有两个自变量 a 和 b，它定义了边长为 a 和 b 的直角三角形的斜边长；函数 len_shortstring 则利用标准函数 min 和 len，定义了一个求自变量 x$ 和 y$ 长度较短的字符串的长度。

需要说明的是，函数中的参数并非真正的变量和数组，它们是一个虚拟参量，其意义在于说明自变量的类型、形态，以及在函数运算过程中所扮演的角色。实际上，自变量既没有具体的值，而且系统也未给它们分配存储单元。因此，自变量实际上是形式参数，或者称为哑元。例如，定义 hypotenuse 函数之后，我们能够得到直角三角形斜边的具体值是什么吗?答案是否定的，即根本不可能。其原因在于函数的定义仅确定了一个基于自变量的运算过程。

其次，函数一旦定义，函数名就不能再被定义为变量名和数组名。另外，下面列出的标准函数名不能被标识为自定义函数名，它们是：

DATE、DATE $、TIME、TIME $、EXLINE $、EXTEXT $、EXTYPE、MAXNUM、PI、RND、DET。

自定义函数的执行完全依赖于对它的调用。和标准函数的使用相仿，单行函数的调用也是出现在表达式中，作为表达式的一个运算分量。其调用形式为：

```
〈函数名〉(〈参数1〉[,〈参数2〉]…)
```

其中，〈参数1〉[,〈参数2〉]…称为实际参数表。实际参数(简称实参)是使用函数时需要代入形式参数的具体内容，其形态可以是常量，简单变量，数组元素，数组名，表达式，甚至是自定义函数名。但应注意，调用函数时，实际参数与形式参数必须保持类型、形态和个数上完全对应一致。对无参函数的调用，仅给出函数名即可，而不应该再添加实际参数。

可见，对自定义函数的调用是很灵活的。例如，对函数 len_shortstring 可以有多种调用形式：

```
PRINT len_shortstring("Wang",name$)
PRINT len_shortstring(city$(i),city$(j))
PRINT len_shortstring(str$(a-(b*c)),d$ & "69")
```

等等。只要实际参数符合形式参数所要求的类型、形态和个数即可。

调用自定义函数的执行过程是：首先开辟若干个临时存储单元(其个数与形参个数相等)，如果实际参数是表达式，则进行计算，然后将实际参数的值对应传送到临时存储单元中，并按照所定义的函数运算关系进行运算，其结果作为该次调用的函数值，返回到调用它的表达式中。最后，释放调用函数时所开辟的临时存储单元。

可以看出，调用函数时实际参数与形式参数之间是按值单向传递的，它们之间是实与虚的

结合。鉴此，在定义函数时，形式参数取什么名字无关紧要，即便与实际参数同名也无妨。

〔例7-1〕 求一元二次方程 $ax^2+bx+c=0$的实根。

程序为：

```
! Find the real roots of an equation
DEF delta(x,y,z)=y * y-4 * x * z

DO
     INPUT prompt "Enter a,b,c:":a,b,c
     IF a=0 then
        PRINT "a=0!"
        PRINT "x=";-c/b
     ELSE
        SELECT CASE delta(a,b,c)
        CASE is>0
              LET x1=(-b+sqr(delta(a,b,c)))/(2 * a)
              LET x2=(-b-sqr(delta(a,b,c)))/(2 * a)
              PRINT "x1=";x1,"x2=";x2
        CASE is=0
              LET x=(-1) * b/(2 * a)
              PRINT "x1=x2=";x
        CASE else
              PRINT "This equation hav't real roots"
        END SELECT
     END IF
     INPUT prompt "continue (Y/N)":ans $
LOOP until lcase $ (ans $ )="n"
END
```

程序中将表达式 y * y－4 * x * z 定义为内部函数 delta，x、y、z 是它的形式参数。程序中共有三处调用此函数，且调用形式相同。调用时，将实际参数 a、b、c 的值对应传送给为形参 x、y、z 所开辟的临时存储单元中，经运算后，返回一个数值到程序调用处，并释放本次调用函数时开辟的临时存储单元。

说明：函数的定义通常应位于使用这一函数的语句之前，否则，必须用 DECLARE DEF（声明）语句事先申明。声明语句的格式为：

```
DECLARE  DEF  〈函数名1〉〔,〈函数名2〉〕…
```

利用该语句对函数进行申明时，仅列出函数名即可，函数名后不能跟形式参数，并且函数名之间用逗号分隔。

2. 多行函数

在一行内无法完成函数运算的定义时，应采用多行函数进行定义。其定义格式为：

```
DEF 〈函数名〉(〈参数1〉[,〈参数2〉]…)
    …
    LET 〈函数名〉=〈表达式〉
    …
END DEF
```

多行函数应以 DEF 开始,以 END DEF 结束,中间包含有若干个语句,用以实现函数运算过程,但其中至少有一个赋值语句,将表达式的值(即函数的运算结果)赋给函数名。

函数中可以使用 EXIT DEF 语句退出该函数模块,但应确保在执行 EXIT DEF 语句退出函数模块之前,先给函数名赋值。

多行函数的调用以及调用函数的执行过程与单行函数完全相同,不再赘述。

〔**例7-2**〕 定义一个求 n 阶乘函数,并利用它计算组合 C_m^n。

根据数学知识我们知道:$C_m^n=\frac{m!}{n!(m-n)!}$

程序为:

```
! Calculate the combination of size n in size m
DEF fact(n)
    LET fac=1
    IF n=1 or n=0 then
       LET fact=1
    ELSE
       FOR i=n to 2 step -1
           LET fac=fac * i
       NEXT i
       LET fact=fac
    END IF
END DEF

! Main program
INPUT prompt "Enter m,n:":m,n
LET com=fact(m)/(fact(n) * fact(m-n))
PRINT "Combination is";com
END
```

在多行函数 fact 的定义中,首先对参数 n 进行测试,如果 n 等于1或者等于0,则直接将1赋给函数名,作为函数的返回值,否则,利用 FOR 循环计算 n 的阶乘,然后将运算结果传送给函数名,作为函数的返回值。

请读者为变量 m,n 任意输入两个正整数,比如3和2,分析一下程序是如何通过调用函数来计算组合数的。

〔**例7-3**〕 定义一个字符串处理函数,将字符串中的字符按逆向顺序重新排列。并利用该函数实现十进制整数转换为二进制数。

程序为：

```
! Define the reverse $ function
DEF reverse $ (s $ )
    LET result $ =“”
    FOR i=len(s $ ) to 1 step -1
        LET result $ =result $ & s $ [i:i]
    NEXT i
    LET reverse $ =result $
END DEF

! Main program
DO
    INPUT prompt “Enter an integer:”:n
    LET n1=abs(n)
    LET string $ =“”
    DO while n1<>0
        LET string $ =string $ & str $ (mod(n1,2))
        LET n1=int(n1/2)
    LOOP
    PRINT “(”;str $ (n);“)10=(”;
    IF n<0 then PRINT “-”;
    PRINT reverse $ (string $ );“)2”
    INPUT prompt “continue (y/n)?”:ans $
LOOP until lcase $ (ans $ )=“n”
END
```

〔**例7-4**〕 定义一个求一维数组各元素之和的函数，调用该函数计算某班级全部学生的平均年龄。

程序为：

```
! Calculate the sum of all elements in array
DEF sum(a())
    LET s=0
    FOR i=lbound(a) to ubound(a)
        LET s=s+a(i)
    NEXT i
    LET sum=s
END DEF
! Main program
DIM age(10)
MAT INPUT prompt “Enter all ages:”:age(?)
```

```
PRINT "The arerage age of all students is";
PRINT round(sum(age())/size(age).1)
END
```

此程序说明了数组作为函数的参数时函数的定义和调用形式。True BASIC 规定，如果形式参数是数组，数组名应后接一对圆括号，并在括号内用逗号说明数组的维数。例如，a()，b(,)，c$(,,)分别表示是一维、二维的数值型数组和三维的字符串型数组。调用函数时，与数组形参相对应的实际参数必须是类型相同维数相同的数组名，True BASIC 2.03版还允许用圆括号和逗号标明数组的维数。但读者应注意，如果使用 True BASIC 1.0版本时，实际数组只能用数组名，而不能用圆括号和逗号标明数组的维数。

程序中，函数 round(sum(age())/size(age),1)的求值过程是：首先计算 sum(age())，即调用多行函数 sum，计算全部学生的年龄之和，然后通过使用数组测试函数 size(age)求数组 age 的大小，即输入数据的个数，最后计算平均年龄，并且将计算结果的小数点后第二位作四舍五入处理。

3. 使用内部函数应注意事项

(1) 定义一个内部函数之后，可以在同一个程序单位中多次调用。每调用一次返回一个值。如果定义的是数值型函数，返回一个数值；定义的是字符串型函数，则返回一个字符串。

(2) 函数过程定义时使用的形式参数只是给出了它在函数计算过程中所扮演的角色，而真正的演员是调用函数时所给出的实际参数。形参可能会与程序其他部分中的变量同名，但它们互不相干。

(3) 函数的计算结果靠函数名返回到调用它的表达式中，而不是靠形式参数返回函数值。在调用函数的执行过程中，形式参数的值即便发生变化，也不影响与该形参对应的实际参数的值。

〔**例7-5**〕 实参与形参的结合关系示例。

```
DEF f(n)
    LET n=5
    LET f=n*n
END DEF
! Main program
LET n=10
LET m=20
PRINT "f(m)=";f(m)
PRINT "n=";n,"m=";m
END
```

———————————————————— ex07-05 ————————————————

```
OK. run
f(m)= 25
n=10          m= 20
```

由运行结果可以看出，调用内部函数过程中，形参 n 的值发生改变，但函数运算结束返回

调用程序后，调用程序中的n未受影响，它仍保留原有值，即等于10，并且也未影响实际参数m的值。换言之，调用程序中的变量n虽然与内部函数中的形参n同名，但它们之间毫无关联。此外，实参m与形参n的结合是单向值传递，函数运算结束后，函数值等于25，而不会再将形参n的值向实参m作反方向值传递。

(4)在内部函数中，除了形式参数外，有时还要使用一些其他变量，这些量是“全局变量”，它们既在函数范围内有效，也在整个程序单位的范围内有效。用户在定义内部函数时，必须慎重地使用全局变量，稍不注意，就会出现意想不到的错误。

〔**例7-6**〕 含有错误的程序示例。

```
! There is an error following the program
DEF fact(n)
    LET fac=1
    IF n=1 or n=0 then
        LET fact=1
    ELSE
        FOR i=2 to n
            LET fac=fac * i
        NEXT i
        LET fact=fac
    END IF
END DEF
! Main program
FOR i=1 to 4
    INPUT prompt "Enter n:":n
    PRINT n;"!=";fact(n)
NEXT i
END
```

程序的本意是计算4个数的阶乘，然而运行程序时如果输入5，则显示“5 != 120”之后就会结束程序的运行。为什么会出现这一错误呢？原因是函数fact中使用的变量i是全局变量。执行调用程序中的FOR循环时，i的初始值等于1，以实参5调用fact函数后，i值变为6。返回主程序后，因为i的值已超过循环终限值4，于是便结束循环，即仅计算一次阶乘运算。

在内部函数中使用全局变量，必须认真考虑其值在函数内外共享的问题，否则便会埋下祸根。为避免这类错误的发生，最好使用外部函数。

(5)在内部函数中，不允许再定义函数，即不允许出现内部函数在物理位置上的嵌套。而在内部函数运算过程中，可以调用同属于该程序单位的其他内部函数或子程序，也可以调用外部函数或外部子程序，甚至可以调用自身。但不能调用其他程序单位中的内部函数或内部子程序。

(6)内部函数的形式参数不能是通道。关于通道的概念将在第九章中讲述。

二、外部函数

外部函数本身构成一个程序单位。它应定义在主程序的 END 语句之后，或者在库中进行定义。外部函数的定义格式和调用格式与内部函数相同，既可以定义单行函数，也可以定义多行函数；可以定义为无参函数，也可以定义为有参函数；函数运算过程中可以包括 EXIT DEF 语句；调用时也将返回一个值，但应注意，在调用程序中必须在调用外部函数之前，用 DECLARE DEF 语句声明。

使用外部函数时应注意事项：

(1) 外部函数中的变量均为局部变量，即使这些变量与主程序或者调用程序中的某些变量同名，调用函数时，它们之间也不会发生干扰。并且每次调用外部函数时，这些量将被重新初始化，即数值型变量初始化为0，字符串型变量初始化为空串。

(2) 外部函数中的数据区和数据指针也是局部的。READ 或 MAT READ 语句只能从所处的程序单位的数据区中读取数据，并且 RESTORE 语句只能恢复该程序单位中的数据指针。

(3) 根据需要，在外部函数中可以定义隶属于它的内部函数。在外部函数运算过程中，可以调用其内部函数，其他外部函数或者外部子程序，甚至可以调用该函数自身。但不能调用主程序或其他程序单位中的内部函数或内部子程序。

(4) 通道不能作为外部函数的形式参数。但在外部函数运算过程中可以打开文件通道，而这种通道的作用域仅限于该函数中，即局部通道。当从外部函数返回调用程序时，无论是否执行 CLOSE 语句，系统都将关闭通道。

〔**例7-7**〕 某企业对产品产量进行统计分析，每种产品包括：产品代码，产品名称，1～12月的产量。要求编制程序对每一种产品全年的生产数据进行处理，找出最大月产量和相应的月份。

分析：设有 n 种产品，将产品代码和产品名称组织成 n×2的字符串型数组 product $，1～12月的产量组织成 n×12的数值型数组 quan。为找出每一种产品的最大月产量和相应的月份，另外定义一个含12个元素的一维数组 max_quan，并定义一个外部函数 lacation_maximum，计算一维数值型数组 array 中的最大值所处的位置，通过调用该函数，找出一种产品的最大月产量和相应的月份，并显示输出。

程序如下：

```
! Date processing program
DECLARE DEF location_maximum
DIM product $ (1,2),quan(1,12),max_quan(12)
INPUT prompt "How many products?":n
MAT redim product $ (n,2),quan(n,12)
FOR i=1 to n
    PRINT "Enter code, name of product";
    INPUT product $ (i,1),product $ (i,2)
    FOR j=1 to 12
        PRINT j;"        month        ouput=";
        INPUT quan(i,j)
    NEXT j
NEXT i
PRINT repeat $ (“   ”,10);“REPORT OF OUTPUT PRODUCT”
PRINT repeat $ (“—”,45)
PRINT “  code      name      month-max      output-max”
PRINT repeat $ (“—”,45)
FOR i=1 to n
    FOR j=1 to 12
        LET max_quan(j)=quan(i,j)
    NEXT j
    LET loc=Location_maximum(max_quan())
    PRINT “   ”;product $ (i,1);“   ”;product $ (i,2);
    PRINT “       ”;loc;“          ”;quan(i,loc)
    PRINT repeat $ (“—”,45)
NEXT i
END

! Find location of maximum from a array
DEF location_maximum(array())
    LET l=lbound(array)
    LET u=ubound(array)
    LET m=array(l)
    LET locat=l
    FOR i=l+1 to u
        IF m<array(i) then
           LET m=array(i)
           LET locat=i
        END IF
    NEXT i
    LET location_maximum=locat
END DEF
```

程序由两个程序单位组成，一个是END语句之前的主程序，另一个是计算数组中的最大值的位置的外部函数。主程序与外部函数之间仅通过数组单向传递数据。外部函数中其余的变量均是局部量，其中的i虽然与主程序中的i同名，但它们之间毫无关联，实际上是不同的变量。

主程序中第二个FOR循环嵌套的j循环，实现第i种产品各月产量传送到max_quan数组对应元素中的操作，然后通过调用函数（实际参数为数组max_quan）求出第i种产品最大月产量的位置，即月份，并输出显示。

三、递归函数程序设计

如前所述，递归是构造算法的一种基本方法。由于很多的数学模型或算法设计方法本来就是递归的，用递归过程来描述它们不仅非常自然，可理解性好，而且证明该算法的正确性比使用非递归形式容易得多。因此，读者应当掌握递归过程的设计方法。

如果递归过程的计算结果是一个单值，可以考虑将此递归过程定义为函数。然而，大多使用外部函数定义递归函数，较少使用内部函数，这是因为内部函数中的变量是全局变量，可能会对递归过程产生干扰，而外部函数中的变量是局部的，对递归过程不存在干扰问题。下面通过两个例题讨论递归函数的定义以及调用过程。

〔**例7-8**〕 阶乘的递归函数定义。

程序如下：

```
! Calculate the combinatorial sum
DECLARE DEF fact
INPUT prompt “Enter m,n:”;m,n
LET con=fact(m)/(fact(n) * fact(m-n))
PRINT “combinatorial sum is”;com
END
! fact function (return n!, recurtive factorial)
DEF fact(n)
    IF n=1 then
       LET fact=1
    ELSE
       LET fact=n * fact(n-1)
    END IF
END DEF
```

其中，外部函数 fact 是按递归形式定义的。为简单起见，下面以5为实际参数值说明递归函数的调用过程。

首先为形式参数 n 开辟存储单元，并将实际参数的值传送到该单位中。执行函数运算时由于形参 n 的值等于5，则条件 n=1的测试值为假，所以执行语句：

LET fact=n * fact(n-1)

先计算表达式的值：5 * fact(5-1)

即：5 * fact(4)

其中又含有函数 fact 的调用，但此时实际参数值等于4。这样依次展开，得到如下结果：

```
fact(5)返回      (5乘
 fact(4)，后者返回     (4乘
  fact(3)，后者返回     (3乘
   fact(2)，后者返回     (2乘
    fact(1)，后者返回     (1)))))
```

递归函数的调用和返回过程如图7-1所示。

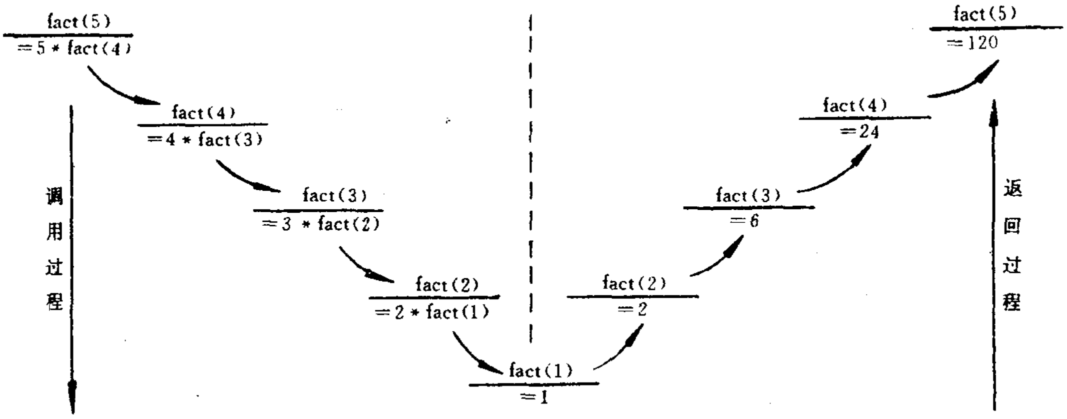

fact(5)
=5 * fact(4)
fact(4)
=4 * fact(3)
fact(3)
=3 * fact(2)
fact(2)
=2 * fact(1)
fact(1)
=1
fact(2)
=2
fact(3)
=6
fact(4)
=24
fact(5)
=120
调用过程
返回过程

图7-1　函数 fact(5)的求解过程

fact 函数的形式参数 n 是虚拟变量，每调用一次 fact 函数就为本次调用的形参 n 开辟一个新的临时存储单元。图7-2表明计算 fact(5)时的存储单元分配记录，框中的数字为存储单元接收的实际参数值，图的右侧标明了返回过程的方向。

从图7-1可知，一个递归的问题可以分为“前推”和“递推”两个阶段。在前推时，fact 函数被调用5次，即 fact(5)，fact(4)，…，fact(1)，此时 fact(1)已知，不必再向前推了。然后开始第二阶段，由 fact(1)推出 fact(2)，由 fact(2)推出 fact(3)，…，一直推算出 fact(5)为止。显而易见，递归过程必须有一个结束递归过程的条件。例如，fact(1)＝1，就是使递归结束的条件。

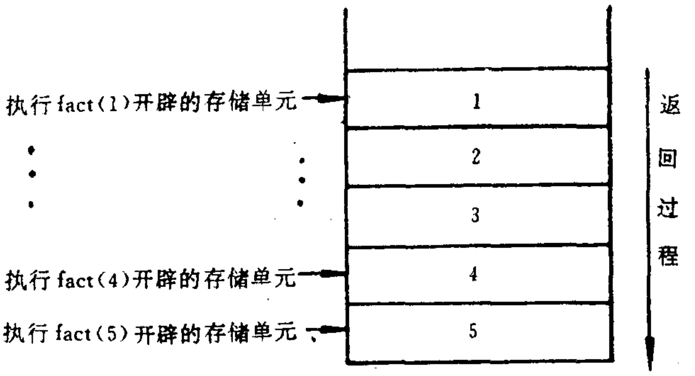

执行 fact(1)开辟的存储单元
执行 fact(4)开辟的存储单元
执行 fact(5)开辟的存储单元
1
2
3
4
5
返回过程

图7-2　计算 fact(5)存储单元分配记录

〔**例7-8**〕　计划生育部门经常需要向政府提供当地未来人口增长趋势。设某城市现有150万人，编程计算今后每年新迁入居民2万人，人口年增长率为 rate，未来若干年后该城市的人口数。

分析：根据当年人口数计算来年人口数的计算公式为：

来年人口数＝(当年人口数)×(1＋rate)＋20000

程序为：

```
! Compute future populations
DECLARE DEF population
```

```
DO
     INPUT prompt "Number of years in future":num_years
     INPUT prompt "Rate of increase is": rate
     PRINT "The population will be";
     PRINT population(num_years,rate)
     PRINT prompt "Continue?(y or n)":answer $
LOOP until lcase $ (answer $ )="n"
END
DEF population(n,r)
     IF n=0 then
         LET population=1500000
     ELSE
         LET population=(1+r) * population(n-1,r)+20000
     END IF
END DEF
```

程序中，population 函数是按递归形式定义的。请读者参照例7-7，分析说明该函数的递归调用过程。

需要补充说明的是，虽然利用递归技术构造算法可使程序简洁易读，容易验证程序的正确性，但是，这是在牺牲存储空间和效率的基础上得到的。因为每层调用都要开辟一组新的临时存储单元，并进行实际参数值向存储单元传递、现场及断点保护等操作。

递归调用分为直接递归和间接递归。如果函数 p 在运算过程中又调用 p，称为直接递归，上面的两个例子就属于这种情况；如果函数 p 在运算过程中调用函数 q，而 q 又反过来调用 p，称为间接递归。当然可以经过多级间接。

按递归方法定义函数时，必须满足以下条件：

(1) 可以把一个问题转化为一个新的问题，而新问题的求解方法仍与原问题的解法相同，只是所处理的对象有所不同，但它们只是有规律地递增或递减。

(2)通过转化过程使问题得到解决。

(3) 必须有结束递归过程的条件。否则递归过程会无限制地进行下去。

第二节　子程序

自定义函数在程序中起着逻辑模块的作用，用于实现一个子问题的求解。但是它的使用具有很大的局限性，原因在于一个函数仅能通过函数名返回一个单值，而无法返回多个值。如果子问题的求解过程要求完成一些操作，或者产生多个结果时，自定义函数就无能为力了。为了解决这类问题，True BASIC 提供了子程序结构。

构造子程序是实现程序部件化的重要手段。与自定义函数一样，子程序在程序中同样起着逻辑模块的作用，并且也分为内部和外部两种，即内部子程序和外部子程序。

一、内部子程序

内部子程序中的“内部”，其含意与前述内部函数中的概念是相同的，即在主程序 END 语句之前或者在其他程序单位内定义的子程序称之为内部子程序。它的定义格式为：

```
SUB 〈子程序名〉(〈参数1〉〔,〈参数2〉〕…)
  ⋮(语句序列,即子程序体)
END SUB
```

其中，SUB 标记子程序的开始，END SUB 标记子程序的结束。子程序名是用户定义的子程序名称，其命名规则与简单变量的命名规则相同，但应注意，它不能用类型说明符“$”结尾，并且不能与调用程序中的变量、函数、数组等同名。

〈参数1〉〔,〈参数2〉〕…是形式参数表，起着调用程序与子程序之间双向传递数据的作用。若调用程序与子程序之间不需要传递数据，则可以不写形式参数表，即可以定义无参子程序。对于有参子程序，形式参数的形态可以是变量名，数组名和通道，而不允许是函数名和子程序名。数组名作为形式参数时，必须用圆括号和逗号指明其维数。

SUB 与 END SUB 之间的语句序列称为子程序体，是实现某一子问题的运算处理过程。其中可以使用 EXIT SUB 语句退出子程序。

SUB、子程序体和 END SUB 构成一个子程序逻辑模块。但是它不能单独运行，必须在调用程序中通过使用 CALL 语句才会被执行。

〔**例7-9**〕 定义一个名为 swap 的子程序，实现二维数值型数组中第 i 行与第 j 行对应元素值进行交换处理。

子程序为：

```
! Swap the value of relevant elements
! in i,j lines of array
SUB swap(array(,),i,j)
     LET l=lbound(array,2)
     LET u=ubound(array,2)
     FOR k=l to u
          LET t=array(i,k)
          LET array(i,k)=array(j,k)
          LET array(j,k)=t
     NEXT k
END SUB
```

子程序中共设置了三个形式参数，其中 array 是一个数组，且标明是数值型二维数组，i 和 j 是数值型变量。实际上，子程序中的形式参数也是一种虚拟变量，“形式上”起着变量的作用，用来说明所传递数据的类型、形态以及在实现特定问题的运算中扮演的角色。只要类型和形态符合要求，形式参数取什么名字无关紧要。比如，子程序 subroutine 为实现某一特定运算，需要有三个参数，第一个为数值型简单变量，第二个为字符串型变量，第三个为数值型二维数组，如下 SUB 行的书写都是正确的：

```
SUB subroutine(a,b$,c(,))
SUB subroutine(var,string$,array(,))
```

子程序中的形式参数，其作用与自定义函数中的形式参数有明显区别，它既接收自调用程序向子程序传递的数据，也将把子程序运算的结果传递给调用程序。换言之，如果子程序的运算依赖于某个形式参数所接收的值，那么这个形式参数起着输入数据的作用；如果子程序运行后某个形式参数取得了新值，那么新值将回送给对应的实际参数，这个形式参数便起着输出数据的作用。

显而易见，上述子程序 swap 的形式参数 i 和 j，其作用是指出需要对哪两行中的元素值进行对应交换，它们具有输入数据的性质，因此，可以称 i 和 j 是输入参数。数组参数 array 也具有输入数据的性质，但是执行子程序后，该数组中的第 i 行与第 j 行对应元素的值进行了交换，所以数组参数 array 是一个输入输出参数。

与自定义函数的调用形式不同，子程序的调用使用 CALL 语句。其语句格式为：

```
CALL 〈子程序名〉(〈参数1〉[,〈参数2〉]…)
```

格式中的参数称为实际参数。它可以是常量、表达式、变量、数组或通道号。与自定义函数相仿，数组作为实参时，True BASIC 2.03允许其后用圆括号和逗号标明其维数，而 True BASIC 1.0版规定数组名后不能加圆括号和逗号。

CALL 语句中的实际参数应当与子程序形式参数的类型、形态（如变量、数组、通道）对应一致，个数相等。例如，对子程序 swap 可以有多种调用形式：

```
CALL swap(score(,),5,6)
CALL swap(score(,),i,n)
CALL swap(quan(,),i,j)
```

等等。但是，如下各种调用则是错误的：

```
CALL swap(5,6,score(,))
CALL swap(t$(,),i,j)
CALL swap(a(),5,6)
CALL swap(score(,))
```

又如，子程序 subroutine 的 SUB 行按如下方式定义：

```
SUB suboutine(a,b$,c(,),#999)
```

对该子程序可以有多种调用形式，如：

```
CALL subroutine(var,string$,array(,),#4)
CALL subroutine(5,name$(i),t(,),#999)
CALL subroutine(x+y,d$ & lcase$(e$),c(,),#1)
```

该子程序中使用了通道号#999作为形式参数，通道参数要写成一个“#”后跟一个1～999之间的任意整常数，而不能写成类似#n 的形式，调用子程序时，将用实际的通道号来代替，如#4，#999，#1。第九章将对通道作更为详细的介绍，在此读者不必细究。

读者应该注意的是，若子程序中某个形式参数起着将运算结果返回到调用程序的作用，则与其对应的实际参数不能是常量或表达式，只能是变量名或数组名。

为了便于理解调用子程序的执行过程，调用程序与子程序之间如何传递数据，下面给出一

个演示程序。

〔**例7-10**〕 调用程序与子程序之间传递数据的演示程序。

```
! Demonstrate program
! Subroutine
SUB add1(x)
     LET x=x+1
     PRINT "x=";x,
END SUB

! Main program
LET a=20
CALL add1(a+10)
PRINT "a=";a,
CALL add1(a)
PRINT "a=";a
END
```

——————————————————ex07-10——————————————————

```
OK. run
x= 31        a= 20        x= 21        a= 21
```

程序说明：第一次调用子程序时，实际参数是一个表达式(a+10)，计算后将结果传递给形式参数 x，经运算之后，输出“x=31”，返回主程序后执行第一个 PRINT 语句，输出“a=20”，即执行子程序后并未改变主程序中 a 的值。第二次调用子程序时，实际参数是变量 a，此时将与变量 a 对应的存储单元的地址传送给形参，也就是在子程序中对形参的引用和赋值实质上变为对实参变量的引用和赋值。所以在子程序中执行 LET 语句之后，输出“x=21”，返回主程序后 a 的值也发生了改变，即执行第二个 PRINT 语句输出“a=21”。

调用子程序的执行过程是，系统自动转向 CALL 语句所给子程序中的定义模块，执行子程序中的语句序列，遇到 END SUB 时，由子程序返回到调用程序中的 CALL 语句后，继续执行后续语句。如果在执行子程序语句序列中遇有 EXIT SUB 语句，可直接中止子程序的执行，提前返回调用程序。若调用带参子程序，则在执行子程序语句序列之前，对实参和形参尚需进行值传送和地址传送操作，实现调用程序与子程序之间的数据传递。

如果实际参数是常量或表达式，实参和形参之间按值传送规则传递数据。此时，首先为形式参数开辟临时存储单元，并把常量或表达式的值传送到这个存储单元中。这样，在执行子程序体的过程中，涉及该形式参数的处理就变为对此存储单元的存取，执行子程序结束后，将释放这个存储单元。因此，子程序中形式参数值的变化，不会影响调用程序中的表达式的值。例如，以 CALL add1(a+10)语句调用子程序后，a 的值不会被改变。由此可知，如果打算把执行子程序后的计算结果返回到调用程序，CALL 语句中的相应实参只能是变量或者数组，而不能是常量或表达式。

若实际参数为变量或数组时，实参与形参之间按地址传送规则进行处理，即把实参的地址

传送给形参，使得子程序体中对该形参的任何引用和赋值都转变为相应实参的引用和赋值。调用一旦结束，实参的值将随形参值的改变而改变。例如，执行 CALL　add1(a)时，实际参数 a 的值为20，经地址传送和计算处理，返回主程序后 a 值改变为21。

鉴于以上讨论可以看出，子程序与自定义函数的调用既有相同之处又有区别。调用程序与子程序之间遵循的值传递规则，与调用自定义函数时的数据传送规则是相同的，都是值单向传送。而自定义函数仅通过函数名将函数运算结果返回调用处，而子程序不是通过子程序名而是通过参数将执行子程序产生的一个或多个运算结果返回调用程序中的，其中的地址传递规则为此提供了保证。

〔例7-11〕 某省计算机学会为加强会员之间的联系，需拟制按姓名字典顺序排列的会员通讯录。设每个会员含姓名，地址，邮政编码，电话号码等项数据，请编程实现。

分析：排序是程序设计中的一种基本运算，为此，设计子程序 sort，实现按第 k 列对字符串型二维数组实施升序排列。

程序为：

```
! Address book program
! Subroutine————→ string array sort
! Array $ (,)          : string array
! k                    : appoint column
! Rule                 :asceuding
SUB sort(array $ (,),k)
    LET l1=lbound(array $ ,1)
    LET u1=ubound(array $ ,1)
    FOR i=u1 to u1-1
        FOR j=i+1 to u1
            IF array $ (i,k)>array $ (j,k) then
               CALL swap(array $ (,),i,j)
            END IF
        NEXT j
    NEXT i
END SUB

! ————————————————————————————————————
! Subroundtine —→ swap values of elements
!                       in i,j lines of array
! Arr $ (,)             :string array
! i,j                   :appoint lines
SUB swap(arr $ (,),i,j)
    LET l2=lbound(arr $ ,2)
    LET u2=ubound(arr $ ,2)
    FOR e=l2 to u2
```

```
            LET t $ =arr $ (i,e)
            LET arr $ (i,e)=arr $ (j,e)
            LET arr $ (j,e)=t $
        NEXT e
    END SUB

    ! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
    ! Main program
    DIM address $ (1,4)
    INPUT prompt “How many member?”:n
    PRiNT “name          address          postcode          telephone”
    MAT INPUT address $ (n,4)
    CALL sort(address $ (,),1)
    PRINT
    PRINT repeat $ (“  ”,20);“ADDRESS  BOOK”
    PRINT repeat $ (“  ”,18);“==============”
    PRINT repeat $ (“-”,56);
    PRINT “  name          address       ”;
    PRINT “   postcode         telephone”
    PRINT repeat $ (“-”,56)
    MAT PRINT address $
    PRINT repeat $ (“-”,56)
    END
```

程序中，共定义了两个内部子程序 sort 和 swap。主程序通过 CALL　sort(address $ (,),1)调用排序子程序，将数组 address $ 按第1列(姓名)实施升序排列。在子程序 sort 语句序列中，通过执行 CALL　swap(array $ (,),i,j)将数组 array $ 中的第 i,j 两行对应元素交换。

上述子程序 sort 仅实现字符串型数组第 k 列升序排列，请读者改写这一子程序，即增添一个表示排列规则要求的形式参数 rule $ ，根据它的不同取值(“ass”代表升序，“des”代表降序)，实现按不同排列要求的排序运算。

定义和调用内部子程序应注意以下事项：

(1) True BASIC 允许子程序多级调用，如上例，这种调用称为逻辑嵌套调用。但定义内部子程序时，其内部不允许再次定义子程序，即内部子程序不允许物理嵌套。此外，内部子程序的调用只能在所处程序单位内发生，并且在内部子程序中，可以调用外部子程序，或者象例7-11那样，调用隶属于同一程序单位的其他内部子程序。甚至允许调用子程序自身。

(2) 与内部函数相仿，内部子程序中引用的除形式参数以外的变量都是全局变量。当内部子程序中的某一变量与所在程序单位中的变量同名时，它们是同一个变量。程序设计中应谨慎使用全局变量，以避免子程序与调用程序之间由全局变量产生的干扰。例如，将例7-11中子程序 swap 内使用的变量 l2，u2改 l1，u1，运行程序时就会产生错误。

换一个角度来讲，就内部子程序而言，子程序与调用程序之间既通过参数传递数据，也可

以通过子程序中出现的全局变量实现数据的传递。此时，全局变量起着输入输出参数的作用。

(3) 如果内部子程序中使用了通道号，该通道也是全局的。若在内部子程序中没有使用CLOSE 语句关闭该通道，则返回调用程序后，可继续使用该通道。

(4) 与自定义函数不同，子程序的定义和调用前后位置无关紧要，即可以先调用，后定义，并且不需要事先声明。

二、外部子程序

外部子程序具有与内部子程序相同的定义格式和调用形式，其主要区别有以下各点：

(1) 外部子程序位于主程序 END 语句之后或建立在库中，即所处的位置与内部子程序不同。

(2) 外部子程序构成一个程序单位，可以在它的内部定义内部子程序、内部函数等。外部子程序可以调用其他外部子程序或外部函数，也可以调用自身。但外部子程序不能调用主程序或其他程序单位中的内部子程序和内部函数。

(3) 外部子程序中所有的变量都是局部变量，即仅在子程序内部起作用。并且每次调用时，这些量都将被重新初始化，即数值量初始化为0，字符量初始化的空串。

外部子程序与调用程序之间仅通过形式参数传递数据，除了形式参数以外，即便使用了与其他程序单位中相同的变量名，也表示不同的变量。

(4) 与外部函数一样，外部子程序中的数据区、数据指针也是局部的。外部子程序中打开的所有通道，即便没有执行 CLOSE 语句进行关闭，返回调用程序时将自动关闭。也就是外部子程序中的通道也是局部的。

使用外部子程序，形式参数是逻辑模块与外界的数据接口，而使用的其他变量、数组、数据区、数据指针、通道等均与外界隔离。因此，构造外部子程序可以确保程序逻辑模块的独立设计，并为其调试、验证和维护带来方便。

〔**例7-12**〕 英文句子中，单词之间用空格分隔。编程从英文句子中把单词分离出来。

程序如下：

```
! Separate word from a sentence
! Main program
DIM word$(1)
PRINT "Enter a senterce:"
LINE INPUT sentence$
CALL separate_word(sentence$,word$())
MAT PRINT word$
END

! ---------------------------------------------------------
! External subroutine
SUB separate_word(l$,w$())
    CALL num_word(l$,n)
    IF n=0 then
    PRINT l$;
    PRINT “ is a null string!”
    EXIT SUB
ELSE
    LET s$=l$ & “ ”
    MAT w$=nul$(n)
    FOR i=1 to n
        LET s$=ltrim$(s$)
        LET w$(i)=s$[1:pos(s$,“ ”)-1]
        LET s$=s$[pos(s$,“ ”):maxnum]
    NEXT i
END IF
END SUB

! ------------------------------------------------------------
! Counting the number of word in a sentence
SUB num_word(line$,num)
    LET num=0
    LET string$=line$ & “ ”
    DO
        LET string$=ltrim$(string$)
        IF len(string$)<>0 then
          LET p=pos(string$,“ ”)
          IF p>0 then
            LET num=num+1
            LET string$=string$[p:maxnum]
          END IF
        ELSE
          EXIT DO
        END IF
    LOOP
END SUB
! ------------------------------------------------------------
```

程序中共定义了两个外部子程序。其中子程序 separate_word 中的形式参数 l$ 用于接收需要分离单词的英文句子，数组 w$ 则用于存放分离出来的各单词，并将数组的值返回到主程序中。执行主程序时，通过 CALL 语句调用子程序 separate_word，进入子程序后，首先调用另一子程序 num_word，以计算字符串中的单词个数(子程序 num_word 中的形式参数 line$ 用于接收调用程序传递来的字符串，即英文句子，形式参数 num 用于存放统计的单词数，并且该值返回到子程序 separate_word)，返回后对 n 值进行测试。如果 n 值等于零，表示输入的英文句子

是空串，显示有关信息后退出子程序，否则，利用 MAT w$ =nul$(n)语句将 w$ 数组的大小调整为 n，并且将调整后的数组各元素赋空串，利用 FOR 循环结构将各单词分离出来并传送给数组 w$ 各元素中。最后返回主程序。

〔例7-13〕 模块化程序设计应用举例。

模块化程序设计是分治策略在程序设计中的应用。当需要处理的问题规模较大时，一般地，按功能将其划分成几个、几十个甚至上百个具有层次关系的功能模块，由顶层或某一层模块规划好下属各模块应实现的目标，以及各模块的数据接口。因为每个模块具有较高的独立性，所以整个程序设计过程可由多名设计人员组成的集体有组织、按计划实施，每人负责一个或几个模块的算法设计，编码和调试，最终再由专门人员将各模块拼装成一个整体。

下面我们以日期转换为例说明上述设计过程。

现在使用的日历称之为格里高利日历，是1582年创立的。试编程实现由数字表示的年、月、日转换成用星期几和用字母拼出的月份所表示的形式。

按照模块化程序设计思想，日期转换程序应包括两个功能：获得日期和显示日期。获得日期模块又包括输入日期和测试输入日期合理性模块，显示日期模块包括变量和数组初始化模块和确定星期几模块。测试日期合理性模块和确定星期几模块还需调用是否是闰年模块。整个程序的结构如图7-3所示。

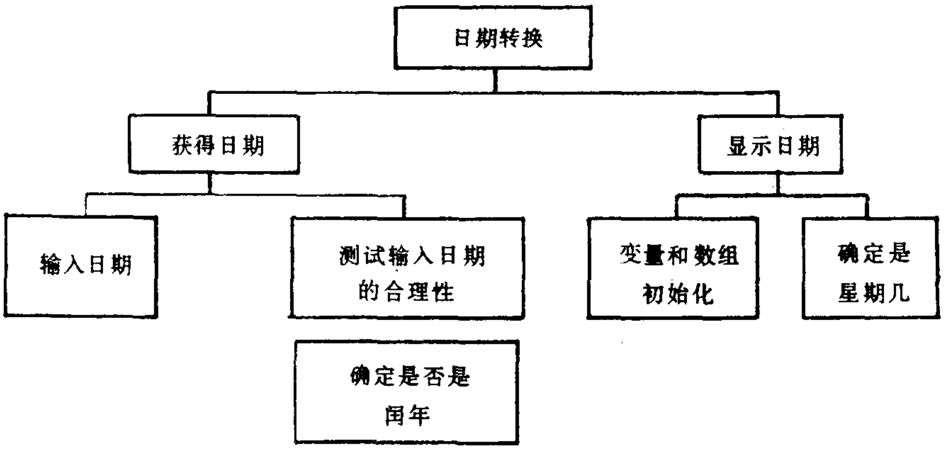


图7-3 日期转换处理程序结构图

程序如下：

```
! Convert numeric date to day of week, year, month, day
CALL getdate(year, month, day)
CALL printdate(year, month, day)
END

! ----------------------------------------------------
SUB getdate(year, month, day)
    DECLARE DEF datevalid_f$
    DO
```

```
            CALL requestdate(year,month, day)
      LOOP until datevalid_f $ (year,month, day)=“t”
END SUB

! ---------------------------------------------------------------
SUB requestdate(year,month, day)
      INPUT prompt “year(yyyy)-------->”:year
      INPUT prompt “month(1-12)------->”:month
      INPUT prompt “day(1-31)-------->”:day
END SUB

! ---------------------------------------------------------------
DEF datevalid_f $ (year,month,day)
      DECLARE DEF leapyear_f $
      LET year_f $ ,month_f $ ,day_f $ =“f”
      IF year>=1 then LET year_f $ =“t”
      IF month>=1 and month<=12 then LET month_f $ =“t”
      SELECT CASE month
      CASE 2
              IF leapyear_f $ (year)=“t” then
                 IF day>=1 and day<=29 then LET day_f $ =“t”
              ELSE
                 IF day>=1 and day<=28 then LET day_f $ =“t”
              END IF
      CASE 4,6,9,11
              IF day>=1 and day<=30 then LET day_f $ =“t”
      CASE else
              IF day>=1 and day<=31 then LET day_f $ =“t”
      END SELECT
      IF year_f $ =“t” and month_f $ =“t” and day_f $ =“t” then
         LET datevalid_f $ =“t”
      ELSE
          LET datevalid_f $ =“f”
          PRINT “Date is not valid”
      END IF
END DEF

! ---------------------------------------------------------------
! Test leapyear
DEF leapyear_f$(year)
    IF mod(year,4)<>0 then
       LET leapyear_f$ =“f”
    ELSE IF mod(year,100)=0 and mod(year,400)<>0 then
        LET leapyear_f$ =“f”
    ELSE
        LET leapyear_f$ =“t”
    END IF
END DEF

! ---------------------------------------------------------
SUB printdate(y,m,d)
    DECLARE DEF day_week
    DIM month$(1:12), day$(0:6)
    CALL initialize(month$(),day$())
    PRINT day$(day_week(y,m,d));“  ”;month$(m);d;“  ”,y
END SUB

! ---------------------------------------------------------
! Array initialize
SUB initialize(m$(),d$())
    MAT READ m$
    MAT READ d$
    DATA January, February, March, April, May, June
    DATA July, August, September
    DATA October, November, December
    DATA Saturday, Sunday, Monday, Tuesday
    DATA Wednesday, Thursday, Friday
END SUB

! ---------------------------------------------------------
! Compute day of week
DEF day_week(year,month, day)
    DECLARE DEF leapyear_f$
    LET total_days=0
    IF month>1 then
       FOR i=1 to month-1
```

```
                READ days
                LET total_days=total_days+days
            NEXT i
        END IF
        LET total_days=total_days+day
        RESTORE
        IF month>2 then
            IF leapyear_f $ (year)="t" then
                LET total_days=total_days+29
            ELSE
                LET total_days=total_days+28
            END IF
        END IF
        LET d1=total_days+year
        LET d2=int((year-1)/4)
        LET d=d1+d2-int((year-1)/100)+int((year-1)/400)
        LET day_week=mod(d,7)
        DATA 31,0,31,30,31,30,31,31,30,31,30
    END DEF
    !----------------------------------------------
```

由于采用多个模块解决问题，所以主程序很短，而且容易理解。子程序 getdate 不仅需要输入年、月、日，而且还要检验输入数据的合理性，由 getdate 所调用的子程序和函数的名字直接反映它们所具有的功能。函数 datevalid_f $ 说明一个函数除了计算一个值以外，可能会做更多的工作，其中一部分即为所给定的年月日不合理时，显示出错误信息。

子程序 printdate 首先定义二个局部数组，其中一个用于存储星期的名字，另一个用于存储月份的名字，然后调用子程序 initialize 向这两个数组赋值，接着通过调用函数 day_week 得到数组 day $ 的下标，以显示当天是星期几的名字。

函数 day_week 首先计算自该年1月1日至当日的总天数，利用公式计算出给定的年月日是星期几。由于排版的原因，计算公式分为三个 LET 语句实现。限于篇幅，该公式的由来不作解释，有兴趣的读者可参阅趣味程序设计书籍。

三、递归子程序设计

上一节讲述了数学函数递归算法的设计方法，其实，递归算法设计策略不仅适用于求解数学问题，在非数值计算领域也得到了广泛的应用。下面举例说明递归策略在数据检索中的应用。

〔例7-14〕 对分查找的递归子程序设计。

对分查找的基本思想已在第六章讲述，这一算法能否用递归方法实现呢？答案是肯定的。

假设一维数组 arr 按升序排列，指针 low 和 high 分别指示待查数据所在范围的下界和上

界，wrd 为待查关键字。对分查询 arr 数组的递归算法可以描述为：首先判断 low 是否小于 high，若条件成立，就计算查找区间中间位置 mid 的值，接着判断 arr(mid)是否等于 wrd，如果相等，说明查找成功，将 mid 的值传送给 pos，作为运算结果并结束递归过程返回调用程序；如果 arr(mid)不等于 wrd，则需要重新执行这一查找过程，即递归调用。但需改变查找区间的上界或下界，即 arr(mid)大于 wrd 时，将继续查找区间的上界修改为 mid－1，下界不变；当 arr(mid)小于等于 wrd 时，将继续查找区间的下界修改为 mid＋1，上界不变。

如果条件“low≤high”不成立，表明查找失败，将0传递给 pos，作为运算结果并结束递归过程，退出子程序。

程序为：

```
! Main program
DIM a(10)
MAT INPUT prompt "Enter your data items:": a(?)
CALL bub_sort(a())
DO
      INPUT prompt "The word to look up is": wrd
      CALL bin_search(a(),lbound(a),ubound(a),p,wrd)
      IF p<>0 then
         PRINT wrd; "is found!"
      ELSE
          PRINT wrd; "Can't be found!"
      END IF
      INPUT prompt "Continue?(y/n)":ans$
LOOP until lcase$(ans$) = "n"
END

! ------------------------------------------------------------
! Bubble sort
SUB bub_sort(arr())
      FOR i=lbound(arr) to ubound(arr)-1
            LET flag=0
            FOR j=lbound(arr) to ubound(arr)-i
                  IF arr(j)>arr(j+1) then
                     CALL swap(arr(j),arr(j+1))
                     LET flag=1
                  END IF
            NEXT j
            IF flag = 0 then EXIT FOR
      NEXT i
END SUB
```

```
! ————————————————————————————————————————
! Swap the values of x and y
SUB swap(x,y)
    LET t=x
    LET x=y
    LET y=t
END SUB

'! ————————————————————————————————————————
! Binary search
SUB bin_search(arr(),low,high, pos, wrd)
    IF low <= high then
      LET mid=int((low+high)/2)
    ELSE
       LET pos=0
       EXIT SUB
    END IF
    IF arr(mid)=wrd then
      LET pos=mid
      EXIT SUB
    ELSE if arr(mid) > wrd then
       CALL bin_search(arr(),low,mid-1,pos, wrd)
    ELSE
       CALL bin_search(arr(),mid+1,high,pos, wrd)
    END IF
END SUB
! ————————————————————————————————————————
```

四、回溯程序设计

需要用计算机解决的问题可以归纳为两类：一类是可以用精确的数学公式描述的问题；另一类是在完成特定的运算过程中，要经过若干个步骤，而每一步都有若干种可能的分支，为了圆满地完成计算，必须遵循一些规则，但这些规则又无法精确地用数学公式或工具来描述。

对于第二类问题，一般采用搜索的方法来解决。回溯就是搜索算法中的一种控制策略。

回溯策略的基本思想已在第一章中介绍，为了使读者更好地理解这一策略，下面讨论八皇后问题的程序设计。

八皇后问题的一个解可以表示为(x_1,…,x_8)，其中 x_i 是把第 i 个皇后放在第 i 行的列数。由于没有两个皇后可以放在同一列上，因此，所有的 x_i 将是截然不同的。例如，(4,6,8,2,7,1,3,5)就是八皇后问题的一个解，即成功布局。其中的4表示 $x_1=4$，即第1个皇后放在第1行的第

4列；6表示 $x_2=6$，即第2个皇后放在第2行的第6列；……。显然所有的 x_i 互不相同。然而如何测试两个皇后是否在同一斜角线上呢？

如果棋盘象二维数组 t(1:8,1:8)那样标记，那么对于在同一条斜角线上的左上方到右下方的每一个元素有相同的“行－列”值，在同一条斜角线上的右上方到左下方的每一元素则有相同的“行＋列”值。若两个皇后处于 t(i,j)和 t(k,l)位置上，那么~~仅当~~

$$i-j=k-l \qquad 或 \qquad i+j=k+l$$

时，它们才处于同一条斜线上。这两个等式可以变换成

$$j-l=i-k \qquad 和 \qquad j-l=k-i$$

因此，当且仅当 $|j-l|=|i-k|$ 时，两个皇后处于同一斜角线上。

定义一个外部函数 place，其功能是：如果第 k 个皇后能放在第 k 行和 x(k)列上，函数值为1，否则为0。其算法可以设计为：将第 k 个皇后与已经放置的第1个到第 k－1个皇后逐一进行测试，判断两个皇后是否在同一列上(其条件为：x(i)＝x(k))，以及是否处于同一斜角线上(其条件为：abs(x(i)－x(k))＝abs(i－k))。比较过程中，一旦发现该皇后与某一皇后产生攻击，则置函数值为0，只有在 k－1次比较结束后，该皇后与前 k－1个皇后均无法相互攻击，表示在第 k 行第 x(k)列上可以放置该皇后，即置函数值为1。

程序为：

```
! ——————————————————————————————————————————
! 8 queens problem
DIM x(1:8)
DECLARE DEF place
LET layout_num=0
LET x(1)=0
LET k=1
DO while k>0
    LET x(k)=x(k)+1
    DO while x(k)<=8 and place(k,x())=0
        LET x(k)=x(k)+1
    LOOP
    IF x(k)<=8 then               ! discover a location
       IF k=8 then                ! discover a layout
          LET string $ =“”
          FOR i=1 to 8
                LET string $ =string $ & str $ (x(i))
          NEXT i
          PRINT using “<##########”:string $ ;
          LET layout_num=layout_num+1
       ELSE
          LET k=k+1
          LET x(k)=0
        END IF
      ELSE
         LET k=k-1                 ! look back upon
      END IF
   LOOP
   PRINT
   PRINT "Layout number is"; layout_num
   END

   ! ------------------------------------------------------------
   ! External function
   ! Place -----→ May place a new queen?
   ! If may place a queen above (k,x(k)) reture 1
   ! else return 0
   DEF place(k,x())
       LET i=1
       DO while i<k
          IF x(i)=x(k) or abs(x(i)-x(k))=abs(i-k) then
             LET place=0
             EXIT DEF
          END IF
          LET i=i+1
       LOOP
       LET place=1
   END DEF
   ! ------------------------------------------------------------
```

说明:

主程序中的 k 和 x(k),分别表示当前行和列,其初始值为1和0。“LET x(k)=x(k)+1”语句表示移到下一列,关系表达式“x(k)<=8 and place (k,x())=0”表示无法在第 k 行第 x(k) 列放置一个皇后,其后的条件“x(k)<=8”表示已找到一个位置,“k=8”则表明已获得一个完整的解。而语句“LET k=k+1”和“LET x(k)=0”表明移向下一行并且置第 x(k)列初始值为0,语句“LET k=k-1”则表明进行回溯。

另外,程序中使用了下一章将要介绍的 PRINT USING 语句。在此,读者可以把它理解为按指定格式显示输出项的值,而不必细究。实际上,执行该语句时,string $ 的值即为八皇后问题的一个解。

运行该程序,将显示输出八皇后问题的92个解。就读者而言,设计该程序可能会有一定的困难,但我们建议你不妨先“走”程序,当你得到问题的第1个解时(其中必然会涉及到回溯过程),对回溯程序设计可能会有更深一层的理解。在此基础上请读者将上述外部函数 place 改写为外部子程序。

八皇后问题有多种求解方法，例如，可以用递归调用过程实现。限于篇幅，在此就不再赘述了。

第三节　程序的链接

当运行程序所需的内存容量超过计算机系统可以提供的内存空间时，程序就无法运行。为此，True BASIC 允许用户将规模较大的程序分解为若干个规模较小的程序，并用 SAVE 命令将它们分别存入磁盘作为 True BASIC 程序文件。运行程序时，利用 CHAIN 语句一个接一个地将磁盘上的小程序装入内存，按顺序执行这些较小的程序。当执行某个小程序时，上一个已被执行过的小程序所占用的内存空间被释放。

利用 CHAIN 语句实现不同程序模块共同享用内存空间，是 True BASIC 解决在小系统与运行大程序的重要手段。

一、CHAIN(链接)语句

CHAIN 语句格式为：

```
   CHAIN "〈文件名〉"
或 CHAIN "〈文件名〉" WITH (〈字符串表达式〉)
或 CHAIN "〈文件名〉",RETURN
或 CHAIN "〈文件名〉" WITH(〈字符串表达式〉),RETURN
```

其中，〈文件名〉是被链接的程序，该程序必须事先编好并用 SAVE 命令存入磁盘。〈字符串表达式〉为内存当前程序传递给被链接程序的参数。

CHAIN 语句功能和使用说明：

(1) 执行前两种格式的 CHAIN 语句时，当前的程序立即停止执行，它所定义和引用的所有变量占用的存储空间将全部释放，所有通道(键盘和显示器占用的＃0通道除外)也被关闭。然后将被链接程序由磁盘装入内存并执行，最终将在调入程序中结束，即不再返回。

(2) 如果使用第二种格式的 CHAIN 语句，被链接程序必须在首行用 PROGRAM 语句列出链接参数。

(3) 执行后两种格式的 CHAIN 语句时，将保留当前程序、占用的存储单元以及当前通道，仅中断当前程序的执行。然后将被链接程序装入内存并运行。执行完后仍返回到原来程序，执行 CHAIN 语句的后继语句。

二、PROGRAM(程序)语句

语句格式为：

```
   PROGRAM  〈程序名〉
或 PROGRAM  〈程序名〉(〈字符串变量〉)
```

其中，〈程序名〉是为该程序标识的名字，其命名规则与简单变量的命名规则相同。〈字符串变量〉为链接参数。

执行该语句时，将接受由前一程序传递过来的参数，并赋给链接参数。若前一程序无参数传递，该语句可以省略，否则 PROGRAM 语句必须置于被链接程序的首行。

写有 PROGRAM 语句的程序可以独立运行，并且该语句将被忽略。注意，含有 CHAIN 语句的程序，不能脱离被链接程序而单独运行。

〔例7-15〕 CHAIN 和 PROGRAM 语句应用示例。

```
! Using chain and program ststements
! Main program
READ string1 $ ,string2 $
PRINT string1 $
CHAIN "b:dis_str" with (string2 $ )
DATA "I am a 90' student"
DATA "I love my motherland"
END

! Chained program
! program name is dis_str
PROGRAM dis_str(str $ )
PRINT str $ & "!"
END
```

请读者根据上述 CHAIN 和 PROGRAM 语句的功能，写出该程序的运行结果。

读者应切记：在执行主程序之前，被链接程序必须事先编好并存入磁盘。

第四节　库

如前所述，外部函数和外部子程序与主程序拼装起来，可以构成一个实现特定运算的可运行程序。但是，函数和子程序一般是作为通用模块来设计的，应该能被实现不同功能的程序所调用。

大型企业通常设有标准件仓库，各分厂或车间组装各种设备时，可以从仓库中领取所需的标准件，以满足生产需要。与此类似，True BASIC 也具有库的功能，这种库建在外存储器（如磁盘）内，库存的“物品”是外部函数、外部子程序以及第10章将要介绍的外部图画。由于它们具有通用性，并且每一个都具有特定的功能，即实现某一特定问题的运算处理，因此，任何程序都可以象领取仓库内的标准件那样，调用库中的函数、子程序或图画，以满足各种处理的需要。

True BASIC 中的库，是外部函数、外部子程序、外部图画的集合，以文件的形式存储于外存中。因而，库亦称为库文件。库不能单独运行，仅供其他程序调用。显而易见，库内的“标准件”越多，开发“产品”的时间开销就越小。

一、如何建立库

库必须以 EXTERNAL（外部）为首语句，其后放置若干个函数、子程序或图画（它们之间不能同名），然后用 SAVE 命令将其存入磁盘，这样便建立了一个库（文件）。

EXTERNAL 语句的作用是声明库中的程序单位都是外部模块，可供其他程序调用。其语句格式为：

```
EXTERNAL
```

外部函数、子程序和图画置入库中以后，就成为公用的程序部件，因而库的作用是为其他程序提供实现某一特定功能的逻辑模块，但应注意，库不能包含主程序，不能以 END 语句结束。库不能独立运行。

〔**例7-16**〕 建立一个名为 strplib. TRU 的库，其中置入本章已给出的函数 reverse $ 以及子程序 separate_word 和 num_words，以供其他程序调用。

受篇幅限制，下面仅列出该库的结构。

```
! Library file
! include 1 function and 2 subroutines

EXTERNAL

! Reture a string of arrange in reverse order
DEF reverse $ (s $ )
    ...
    ...
    ...
END DEF

! Separate words from a string
SUB separate_word(l $ ,w $ ())
    ...
    ...
    ...
END SUB

! Count the number of words in a string
SUB num_words(line $ ,num)
    ...
    ...
    ...
END SUB
```

二、如何使用库

建立库之后，任何程序均可调用库中的逻辑模块。但在调用程序中必须使用 LIBRARY（库）语句指定所用库的文件名。该语句格式为：

```
LIBRARY  "〈库文件名1〉"〔,"〈库文件名2〉"〕…
```

如果需要调用库中的某个函数，除了使用 LIBRARY 语句外，还要在调用程序中用 DECLARE DEF 语句声明所要使用的函数。对库中子程序的调用则不存在此限制。

需要补充说明的是，库中各程序单位之间同样允许相互调用，但应注意，调用函数模块时，该程序单位内仍需要用 DECLARE DEF 语句进行声明。

〔**例7-17**〕 设字符串变量 stockmessage $ 的值由多项字符串型数据连接而成，各项数据之间用一个或多个空格分隔。利用库 strplib. TRU 分解该字符串变量的值，以便获得各项数据，并对其进行处理。

程序片段为：

```
! Use of Library
LIBRARY "B:strplib. TRU"
DIM arr $ (1)
    …
    …
    …
INPUT prompt "Enter message of stock:";stockmessage $
CALL separate_word(stockmessage $ ,arr $ ())
    …
    …
    …
END
```

程序中仅列出 LIBRARY 语句以及与分解字符串变量 stockmessage $ 有关的语句，其他处理过程省略。

第五节 模 块

程序设计中，人们通常利用外部的函数、子程序构造程序部件，但由于除了形式参数以外的所有量都是局部的，因此也会遇到一些问题。例如，如果与调用程序传递的数据很多，势必需要设置过多的形式参数。这显然与程序设计心理学家的研究成果相违背，即程序逻辑模块的参数数目应限制在10个以内，超出这个限制，就极容易产生错误。又如，同一程序部件的两次调用之间(或互不调用的两个程序部件之间)无法直接传递数据，而只能借助于主程序来传递。

为了解决这类问题，True BASIC2. 03新设了一种构造程序部件的结构，即模块。在这种结构内可以通过定义公用变量和共享变量来解决上述问题。所谓公用变量是指能够被主程序和所有外部函数、外部子程序共同使用。共享变量与公用变量不同，它只能被相关的外部函数和外部子程序共同使用。

一、如何定义模块

这里所说的模块不是程序设计理论中称谓的程序逻辑模块，而是 True BASIC 2. 03提供的一种新的程序结构。它是外部函数、外部子程序的集合，既可以在主程序 END 语句之后进行定义，也可以在库中进行定义。模块的作用是将彼此相关的外部函数、外部子程序拼装在一

起，通过指定公用变量、共享变量或者私有函数或子程序，以解决形式参数设置过多以及借助于主程序或调用程序传递本来与它无关的数据问题。模块的定义格式为：

```
MODULE 〈模块名〉
    PUBLIC(公用)语句部分
    SHARE(共享)语句部分
    模块初始化部分
    外部函数和外部子程序部分
END MODULE
```

模块的定义以 MODULE 开始，以 END　MODULE 结束。〈模块名〉是用户为模块标识的名字，其命名规则与简单变量名的命名规则相同。

PUBLIC(公用)语句部分由一个或多个 PUBLIC 语句组成。其作用是用来定义模块内所有的函数和子程序中哪些量是公用变量。注意，用该语句定义数组时，必须给出数组各维的上下界。例如：

PUBLIC s,t $,u(－10:10)

公用变量的作用域既包括置入模块内的函数和子程序，也包括主程序和所有外部函数和外部子程序。例如，模块内一子程序 subr 中的变量 x 被定义为公用变量后，当由调用程序(可能是主程序)调用子程序 subr 并返回时，x 的值并不消失而保留原有值，即 x 占用的存储单元并不释放，若重新引用该变量，x 的值就是上一次调用子程序 subr 结束时的值。显然，子程序 subr 中的 x 变量的作用域已不是构造模块之前的局部量，而成为所有程序单位中均能使用的量。

SHARE(共享)语句部分由一个或多个 SHARE 语句组成。其作用是用来定义哪些变量在模块内是共享变量。与 PUBLIC 语句相仿，用 SHARE 语句定义数组时，也要标明数组各维的上下界。例如

SHARE l,m $,n(100)

共享变量的作用域与公用变量的作用域截然不同，它仅在模块内部中的所有子程序和函数范围内有效，在作用域以外，便失去共享特性。

PRIVATE(私有)语句部分由一个或多个 PRIVATE 语句组成，其作用是用来说明模块内哪些子程序和函数是私有的。被指定为私有的函数和子程序只能被模块内的程序单位来调用，而不允许主程序或模块外的程序单位所调用，这也就是“私有”的含义。

如果模块内的函数被说明为私有的，调用程序单位中不需要再用 DECLARE　DEF 语句进行声明。

书写 PRIVATE 语句时，仅列出函数名或子程序名即可。

模块初始化部分的作用是通过若干个语句为指定的公用变量和共享变量提供初始值。该部分应位于 PUBLIC、SHARE 和 PRIVATE 语句之后，外部函数和外部子程序之前。

外部函数和子程序定义部分，是所有置入模块的函数和子程序。

说明：(1) 模块定义格式中的前四部分都是可选的，即根据需要可以选取其中几部分。

(2) 当模块内某个程序单位仅与模块内的其他程序单位有关联，而与模块外界无关时，该程序单位才能被指定为是私有的，其作用相当于将该程序单位屏蔽在模块内部。

〔**例7-18**〕 设计一个模块，实现堆栈的基本操作。

堆栈是程序设计中常用的一种数据结构。它的存储原理如同一个子弹夹，子弹只能从子弹夹的顶部压入或弹出。子弹进出子弹夹遵循“后进先出”或“先进后出”的原则。具体地讲，堆栈是在存储区中开辟的一个连续的工作空间，也称为栈表。栈表的一端固定，称为栈底，另一端浮动，称为栈顶。栈中的元素只能由栈顶进出堆栈，并遵循“后进先出”的原则。也就是说，只能在栈顶位置进行压入(push) 或弹出(pop)操作。

为了实现堆栈的操作，需要设计以下子程序：

inistack(n)　堆栈初始化，并将堆栈尺寸(所能存储的数据个数)设置为 n；

push(x)　压栈操作，即将变量 x 的值压入栈内所有数据的最上层，使之成为栈顶元素；

pop(x)　出栈操作，即将栈顶元素弹出并赋给变量 x。

另外，设置变量 Freestack，反映堆栈存储数据时，剩余空间还能够存储数据的个数。

将有关堆栈的操作定义在模块 stack 内，然后将模块存放在库“stack. TRU”中。

```
EXTERNAL
MODULE stack
        PUBLIC freestack
        SHARE stack(100),stacksize,top
        PRIVATE stackerror

        ! Module initial
        LET top=0              ! Top is pointer of stack
        LET stacksize, freestack=100

        ! Stack initial
        SUB inistack(n)
              LET stacksize=n
              MAT stack=zer(n)
        END SUB

        ! Push a element onto stack
        SUB push(x)
              IF top<stacksize then
                 LET top=top+1
                 LET freestack=stacksize-top
                 LET stack(top)=x
              ELSE
                  CALL stackerror
              END IF
        END SUB

        ! Pop a element from stack
        SUB pop(x)
            IF top>0 then
               LET x=stack(top)
               LET top=top-1
               LET freestack=stacksize-top
            ELSE
                CALL stackerror
            END IF
        END SUB

        ! Error processing
        SUB stackerror
            IF top=stacksize then
               PRINT "stack overflow!"
            ELSE
                PRINT "stack empty!"
            END IF
        END SUB
END MODULE
```

在模块 stack 中，freestack 变量被定义为公用变量，因此，从模块外界就可以引用它的值，即了解堆栈的剩余空间。变量 stacksize（堆栈尺寸），top（指针）和数组 stack（堆栈）被定义为模块的共享变量，它们的作用域仅局限于模块内部，不会受到模块外界的干扰。子程序 push 和 pop 用来实现堆栈的压栈和出栈操作，而子程序 stackerror 仅在对堆栈作压入、弹出操作出错时，才用来显示出错信息，因此它被定义成模块的私有子程序。经模块初始化后，堆栈指针（top）指向栈底，栈的尺寸定为100。调用 inistack 可以重新设置堆栈的尺寸。

二、如何使用模块

模块本身并不是一个可调用的程序单位，外界只能调用置入模块中的函数和子程序，以及引用被定义为公用变量的变量。调用模块内的程序单位时，应注意：(1) 如果要使用模块内定义的公共变量时，在主程序或模块外的其他程序单位中，必须首先用 DECLARE PUBLIC 语句进行声明，其语句格式为：

DECLARE PUBLIC 〈公用变量1〉〔,〈公用变量2〉〕…

其中，公用变量可以是简单变量，也可以是数组。但对于数组，声明时需要在数组名后用圆括号和逗号标明数组的维数，而不能重新指定数组各维的上下界。

(2) 在模块以外的程序单位中，不能调用模块内的私有函数和子程序。如果调用模块内的非私有函数，仍需要使用 DECLARE DEF 语句进行声明。

〔**例7-19**〕 利用例7-18建立的库“stacklib. TRU”中的 stack 模块，实现十进制整数转化为二进制数。

程序为：

```
! Use of stack (decimal to binary conversion)
LIBRARY "A:stacklib.tru"
DECLARE PUBLIC freestack
INPUT prompt "enter max-length of binary number:":l
CALL inistack(l)
INPUT prompt "Ente a decimal integer:":n
LEt n1=n
DO while freestack>0
    LET bit=mod(n1,2)
    CALL push(bit)
    LET n1=int(n1/2)
LOOP until n1=0
PRINT "(";n;")10=(";
DO while freestack<1
    CALL pop(bit)
    PRINT str$(bit);
LOOP
PRINT ")2"
END
```

ex07-19

```
OK. run
enter max-length of binary number:64
Enter a decimal integer: 155
(155)10=(10011011)2
```

习　题

7—1. 首先分析以下各题的特点，然后定义自定义函数，并完成题目所求。

(1) 计算

$$R=\frac{e^{2x}+\sqrt{1+2e^{x}+3e^{2x}}}{25}+\sin^{2}x+\sqrt{1+2\sin x+3\sin^{2}x}$$

其中，x 的值由键盘输入。

(2) 计算

$$y=\frac{1+x}{1-x+x^{2}}\left(\frac{1+Z^{2}}{1-Z^{2}+Z^{4}}+\left(\frac{1+r}{1-r+r^{2}}\right)^{2}\right)$$

7—2. 利用标准函数，编写计算给定实数的任意次幂的自定义函数 power(x,y)，要求能够处理幂指数为实数的情况。然后编写主程序，求给定的两个数之和的 n 次幂与两数

之差的 n 次幂。

提示：$x^y = e^{\ln(x) * y}$

7—3. 定义两个外部函数 a_max(arr()) 和 a_min(arr())，分别求一维数组 arr 的最大值和最小值。然后编写主程序，实现某工厂全年12个月最高月产值和最低月产值的统计处理。

7—4. 编写函数 htoi(n $)，将十六进制数字串 n $（不含小数部分）转换为等价的十进制数。数字串中允许的字符可以是0到9，a 到 f，或者是 A 到 F。

7—5. 编写一个 squeeze $ (s1 $，s2 $) 函数，它从字符串 s1 $ 中删去所有与字符串 s2 $ 中相同的字符。

7—6. 编写 rindex(arr()，key) 函数，它返回在数组 arr 中 key 出现的最右边位置，如果在 arr 中没有与 key 匹配的元素，就返回 −1。

7—7. 编写递归函数，求勒让德多项式的值。已知勒让德多项式为：

$$p_n(x) = \begin{cases} 1 & (n = 0) \\ x & (n = 1) \\ (2n - 1) \cdot x \cdot p_{n-1}(x) - (n - 1) \cdot p_{n-2}(x)/n & (n > 1) \end{cases}$$

输入 x 的值，计算 n=5，8，10时多项式的值。

7—8. 编写递归函数 gcd(x，y)，求任意两个自然数 x 和 y 的最大公约数。

7—9. 编写一子程序 testprime(n，result $)，判断任一自然数 n(n>1) 是否为素数，如果是，result $ 返回值为“y”，否则，返回值为“n”。

7—10. 编写子程序 sort(arr()，rule $)，其中 rule $ 用于接收调用程序的排序规则，若值为“asc”，对数组 arr 按升序排列；若值为“des”，对数组则按降序排列。

7—11. 编写子程序 dictionary(s $，dict $ ())，其中 s $ 表示按行输入的一句话（各单词以空格分隔），要求从 s $ 中找出不相同的所有单词，单词清单以数组 dict $ 返回调用程序。

7—12. 确定下列程序的输出值，并说明理由。

```
(1) DEF f(x)
          IF x * y < 0 then
             LET f=x * y/2
          ELSE
               LET f=sqr(x * y)
          END IF
     END DEF
     ! Main program
     DO
          READ x,y
          PRINT f(x)
     LOOP until x=0
     DATA 10,-3,4,25,-4,-9,0,71
     END
```

```
(2) SUB subt1
        PRINT a;"      ";
        PRINT b;"      ";
    END SUB
    SUB subt2(b)
        PRINT a;"      ";
        PRINT b;"      ";
    END SUB
    ! Main program
    LET a=666
    LET b=888
    FOR i=1 to 3
        CALL subt1
        CALL subt2(a)
        CALL subt3(a+b)
        CALL subt3(i)
        PRINT
    NEXT i
    END
    ! External subroutine
    SUB subt3(a)
        PRINT a;"      ";
        PRINT b;"      ";
    END SUB
```

7—13. 简要说明内部函数与外部函数、内部子程序与外部子程序之间的区别，以及外部函数与外部子程序之间的异同。

7—14. 简述程序链接的概念，并说明在什么情况下用到程序链接技术。

7—15. 建立一个三角函数库（文件），其中需定义的函数有：ctg(x)、sec(x)、cse(x)、arccos(x)、arctg(x)和 arcctg(x)。

7—16. 分析例7-13日期转换程序中的子程序和函数，其中哪些量应作为公用变量或共享变量，哪些程序单位应作为私有函数和子程序，然后将相关的函数和子程序定义为模块。

7—17. 编写三个外部子程序，分别实现十进制整数和小数转换为二进制数，以及实现任一十进制数转换为二进制数。将它们组织成模块，且设置在 binlib. TRU 库中。

第八章　格式化输入与输出

现代程序设计的基本要求是，程序的性能可靠，可读性好，并且具有良好的用户界面。为了满足高标准程序设计要求，True BASIC 提供了功能更强、控制更加灵活的输入输出语句。利用这些语句可以制作出各式各样的输入输出格式，使输入数据的操作更接近于人们的习惯，使输出的数据更清晰、更美观。

本章首先介绍如何通过 SET 语句，tab 函数以及 PRINT　USING 语句实现数据的格式化输出，然后讲述格式化输入中用到的 CLEAR、光标控制以及 GET　KEY 语句，最后讨论菜单驱动程序的设计问题并给出程序设计实例。

第一节　格式化输出

在开发实用程序时，往往对输出格式有很高的要求，仅用 PRINT 和 MAT　PRINT 语句中的分区格式或紧凑格式已无法满足需要。为此，本节将首先介绍如何设置屏幕的行宽和域宽，然后着重讨论输出格式由用户自选的 PRINT　USING 语句。

一、行宽和区宽的设置

在 True BASIC 初始状态下，显示器屏幕的一行设置为 80 个字符宽，一行被均分为五个宽度相等的区域，即每个区域的宽度为 16 个字符。程序设计中，True BASIC 允许用户利用 SET　MARGIN 语句和 SET　ZONEWIDTH 语句重新设置行宽和区宽。其语句格式为：

```
SET  ZONEWIDTH 〈数值表达式〉
SET  MARGIN 〈数值表达式〉
```

说明：(1) 重新设置行宽和区宽时，行宽值必须大于区宽值，并且行宽和区宽只能取大于等于 1 的整数值。

(2) 重新设置行宽和区宽后，PRINT 或 MAT　PRINT 语句按分区格式输出时，将按新行宽和区宽输出。

〔**例 8-1**〕　重新设置行宽和区宽示例。

```
DIM a(10)
SET margin 40
SET zonewidth 8
MAT READ a
MAT PRINT a
DATA 1,2,3,4,5,6,7,8,9,10
END
```

ex08-01

```
OK. run
1             2             3             4             5
6             7             8             9             10
```

输出结果只占整个屏幕的左半部，如果上述程序中不设置行宽，仅将区宽设置为 8，则相当于在屏幕的一行内设置了 10 个区域，运行程序时会使 10 个数据在一行内输出。

(3) 程序中，可以根据实际需要多次设置行宽和区宽。

设计通用程序时，往往需要知道屏幕的当前行宽和区宽值。为此，True BASIC 提供了 ASK MARGIN 和 ASK ZONEWIDTH 两个语句，分别用于实现将屏幕当前的行宽和区宽值送入指定的变量中。其语句格式为：

```
ASK  MARGIN  〈变量〉
ASK  ZONEWIDTH  〈变量〉
```

格式中的变量必须是数值型的。

二、TAB 函数

实际编程时，常常需要将某一输出项从指定位置开始输出。遇到这种情况时，可以在 PRINT 语句中使用 TAB 函数将显示器光标(或打印机字头)移到一个指定位置上。如：

```
PRINT  TAB(5);"red";TAB(15);"yellow";TAB(25);"blue"
```

执行此语句时，字符串“red”将从屏幕当前行第 5 列开始输出，“yellow”从第 15 列开始输出，“blue”将从第 25 列开始输出。

使用 TAB 函数应注意事项：

(1) TAB 函数只能用于 PRINT 语句中。其中的参数可以是一个数值表达式，取值范围为 1～行宽。

(2) TAB 函数与输出项之间应以分号分隔，若用逗号分隔，该 TAB 函数将失去作用，即按分区格式输出输出项的值。

(3) 在一个 PRINT 语句中，可以有多个 TAB 函数，但其中的参数值只能按顺序递增。如果要用 TAB(n)将光标移到当前行的第 n 列，而光标当前位置已超出第 n 列，则系统自动将光标移到下一行的第 n 列位置上。

(4) True BASIC 2.03 中，TAB 函数可以使用两个参数 R(行)和 C(列)，实现屏幕光标任意位置的定位。如：

```
PRINT TAB(5,10);n
```

表示将光标定位于屏幕的第 5 行第 10 列位置，然后输出 n 的值。

如果程序中使用了 TAB(R,C)函数，执行程序时屏幕会自动转到屏幕的文本显示模式状态，显示结束后即停留在此状态。按任一键后，屏幕将返回到编辑窗口和背景窗口状态。

〔**例 8-2**〕利用 TAB 函数制作数据报表。

```
! Making simple report with tab function
PRINT tab(5,9);"Statistical Report"
PRINT repeat$("*",40)
PRINT tab(5);"name";tab(20);"sex";
PRINT tab(27);"age";tab(34);"class"
```

```
    PRINT repeat $ ("*",40)
    DO while more data
        READ name $ ,sex $ ,age,class
        PRINT tab(2);name $ ,tab(20);sex $ ;
        PRINT tab(27);age;tab(34);class
        PRINT repeat $ ("-",40)
    LOOP
    DATA Ji ban gong, man, 19, 7921
    DATA Wang ping, man, 20, 7922
    DATA Li xia, woman, 19, 7914
    DATA Sun wei zhong, man, 21, 7915
    DATA Lin min, woman, 20, 7923
    END
```

执行 RUN 命令后，系统首先清屏幕，然后屏幕转到文本显示模式状态，并从第 5 行开始显示下列报表。

```
                    Statistical Report
    * * * * * * * * * * * * * * * * * * * * * * * * *
       name             sex         age         class
    * * * * * * * * * * * * * * * * * * * * * * * * *
      Ji ban gong       man          19          7921
    - - - - - - - - - - - - - - - - - - - - - - - - -
      Wang ping         man          20          7922
    - - - - - - - - - - - - - - - - - - - - - - - - -
      Li Xia            woman        21          7914
    - - - - - - - - - - - - - - - - - - - - - - - - -
      Sun wei zhong     man          19          7915
    - - - - - - - - - - - - - - - - - - - - - - - - -
      Lin min           woman        20          7923
    - - - - - - - - - - - - - - - - - - - - - - - - -
```

按任一键后，屏幕将恢复到执行 RUN 命令之前的状态。

人们对输出报表的基本要求是数据准确，格式美观整齐。因此，在进行输出报表程序设计之前，必须事先准确地确定表名、栏标题以及每个数据的长度和所处的位置。

三、PRINT USING(自选格式输出)语句

利用 PRINT 语句中的逗号、分号或者 TAB 函数只能确定数据输出的起始位置，而无法确定数据项本身按什么格式输出。例如，确定输出的数据含几位整数，是否有小数部分，若有，小数部分应占几位，钱款数据前是否加币符等等。

利用 PRINT USING 语句可以实现数据本身的格式输出，其中的格式是由用户确定的。它是实现数据按报表形式输出的重要语句，在报表程序设计中得到广泛应用。

1. PRINT USING 语句的格式和功能

语句格式：

```
PRINT USING〈格式串〉:〈数据项1〉,〈数据项2〉,…,〈数据项n〉
或 PRINT USING〈格式串〉:〈数据项1〉,〈数据项2〉,…,〈数据项n〉;
```

其中，"格式串"表明数据项输出时所占的字符宽度以及按什么样的格式输出。格式串中使用的格式字符共有11个，即

< > + − $ * % # , . ∧

按照一定的规则将上述字符进行组合，即可表示出数值型数据或字符串型数据的输出格式。

格式串中除了使用格式字符以外，还可以包含字母字符、空格以及其他字符。执行 PRINT USING 语句时，它们将按照出现在格式串中的原样与输出项一起显示输出。

语句功能：当执行该语句时，将按格式串提供的格式输出各数据项的值。如果格式串中含有非格式字符，则按原样输出。如果语句以分号结束，执行该语句后，光标将停留在输出项的结尾处；如果使用第一种格式，执行语句后光标将移到下一行的起始位置。

利用该语句输出数据项的值时，数据项总是按指定的宽度进行输出，而不受显示区域的限制，数据的精度仅由格式串确定。当输出项的位数超出格式给定的宽度时，将输出一串星号"*"，其个数与格式字符数相等；如果数值型数据的小数部分超过给定宽度时，超出位将四舍五入并被截断。

2. 数值型数据的格式输出

输出数值数据时，格式串可由以下三种格式符组成。

(1) 数值型格式字符

#	每个#号代表一位数字，数据的前置零用空格替换。
*	每个*号代表一位数字，数据的前置零用"*"号替换。
%	每个%号代表一位数字，数据的前置零用"0"(零)表示。

(2) 首部格式字符

在确定的输出格式中，其首部可以是以下符号：

+	在数字前面显示"+"或"−"。
−	在数字前面显示空格(对正数)或"−"(对负数)。
$	在数字前面显示币符"$"。

(3) 其他格式字符

在确定的输出格式中可以插入以下符号：

.	指定小数点的位置。
,	将逗号插入到数字串中适当的位置。
∧	按科学表示法指定输出格式时，它将输出数据的指数部分。

下面通过表格说明数学格式字符的用法。

简单数字格式示例如表8-1。

格式串"####"的含义是输出长度为4的整数。如果输出项整数部分不足4位，用空格替换前置零，如果含有小数，则对其第一位作四舍五入处理。

表8-1　　　　　　　　**简单数字格式示例**

输出项值	格式串	输出结果	说　　明
8	####	␣␣␣8	用空格代替前置零。
-64	####	-␣64	负号在最左面。
1024	####	1024	
-1024	####	****	数据太长，输出一串“*”号。
256.4	####	␣256	对被截断位作四舍五入处理。
512.6	####	␣513	同上。
18	****	**18	用“*”号代替前置零，以防涂改。
-18	****	-*18	
36	%%%%	0036	
-36	%%%%	-036	
94.1	%%%%	0094	

注意，为数值型指定输出格式时，在一个格式串中不能混合使用“#”、“*”、“%”，即它们是互斥的。如“##%%”是不合法的。如果输出项是字符串型数据，这三个格式符可以混合使用。

含有小数的数字格式示例如表8-2。

表8-2　　　　　　　　**含有小数的数字格式示例**

输出项值	格式串	输出结果	说　　明
0.64	###.##	␣␣␣.64	
32	###.##	␣32.00	用零补足小数位。
25.6	###.##	␣25.60	同上。
2.048	###.##	␣␣2.05	对第三位小数四舍五入
1.024	###.##	␣␣1.02	同上。
32	***.**	*32.00	
1243.08	***.**	******	数据太长，输出一串“*”号。
32.129	%%%.%%	032.13	对第三位小数四舍五入。

在一个为数值项输出而确定的格式串中，可以使用一个首部格式字符（+，-，$），也可以连续使用若干个。但对应于小数位的格式字符不允许是首部字符，否则，将视为非法格式串。格式串中如果仅使用一个首部字符，显示数据时，首部字符将作为数据的最左端字符，且位置固

定不变。如果连续使用若干个首部字符，显示数据时，该字符将随数据的大小在指定位内浮动。首部字符和逗号使用示例如表8-3。

表8-3　首部字符和逗号应用示例

输出项值	格式串	输出结果	说明
102.4	++++#.##	␣+102.40	数符位置浮动。
−20.48	----#.##	␣␣−20.48	同上。
64.16	$$$$#.##	␣␣$64.16	“$”位置浮动。
−12.8	$$$$#.##	␣$−12.80	同上。
64	##,###.##	␣␣␣␣64.00	逗号位置用空格替换。
198.2	$$,$$#.##	␣␣$198.20	逗号位置用“$”替换。
1836.46	$$,$$#.##	$1,836.46	
−0.6	+++++.##	␣␣␣␣−.60	
−0.8	-----.##	␣␣␣␣−.80	
0.456	+++++.++	——	非法格式串。
0	-----.--	——	同上。
0	$$$$$:$$	——	同上。

科学表示法格式示例如表8-4。

表8-4　科学表示法格式示例

输出项值	格式串	输出结果	说　　明
4.2e3	#.#∧∧∧∧	4.2e+03	指数部分占4位，含“e+”或“e−”。
4.7e3	##.#∧∧∧∧	47.0e+02	同上。

注意事项：

（1）一个格式串中至少要有一个数字格式符，否则，系统将视为非法格式串。在一个格式串中，根据需要可以定义两个或更多的输出格式。此时，格式之间必须有非格式字符（包括空格）分隔。如，执行

PRINT　USING “##␣plus␣##␣=###”:n1,n2,s

时，若变量n1、n2、s的值分别为80,70,150，则输出结果为：80␣plus␣70␣=150。

若要求输出的两个数据之间同时输出一个逗号，可以使用如下形式的PRINT　USING语句：

PRINT　USING “##␣,␣##”:n1,n2

设n1、n2分别取值为10,20，输出结果为：10␣,␣20。这种输出格式在建立正文文件时是很有用的。

（2）在一个PRINT　USING语句中，如果输出项多于格式串中的格式数，多出的输出项从最左端开始重新使用格式项；反之，剩余的格式项作废。但是格式串中的非格式符总是要输

出的。例如

PRINT　USING“###␣,␣$$#,###.##”:n1,n2,n3

格式串中共定义了两个输出格式，因此，输出项n3使用的格式为“###”。

3.字符串的格式输出

输出字符串时，其格式串可以由任意数值型格式符来定义。当输出项的长度小于格式串指定的长度时，字符串显示在中间位置，两侧由空格填充，空格总数为奇数时，左侧的空格比右侧的空格少一个。如果字符串的长度大于格式串指定的长度时，输出一串“*”号，其个数与格式串的长度相等。例如：

PRINT USING“########”:“pen”　　输出结果　“␣␣pen␣␣␣”

PRINT USING“%%%%%%%%”:“book”　　输出结果　“␣␣book␣␣”

PRINT USING“********”:“pencil”　　输出结果　“␣pencil␣”

PRINT USING“###.##”:“nextbook”　　输出结果　“******”

与数值项格式输出的不同之处在于，输出字符串时，格式字符#、*、%可以在格式串中混合使用。另外，在格式串前可以加首部格式符“<”或“>”，此时，字符串将靠左端或右端对齐输出。如，执行

PRINT using“<####################”:name $

时，将占据21个字符宽度，并且按左对齐原则输出变量name $ 的值。注意，格式字符“<”或“>”也要占据一个字符宽度。

如果PRINT　USING语句的格式串其长度较长，可以事先将格式串赋给一个字符串型变量，然后在语句中直接引用该变量作为数据的输出格式。

四、MAT　PRINT　USING语句

该语句实际是MAT　PRINT语句和PRINT　USING语句的组合，其功能是按格式串中指定的格式输出数组所有元素的值。如果格式串中包含多个格式项，则总是按指定的格式输出数组的部分元素。语句格式为：

```
MAT PRINT USING〈格式串〉:〈数组1〉,〈数组2〉,…
```

使用该语句时，通常需要对输出数组各列元素的值进行分析。比如，各列数据的长度不超过4位，且是整数，可以选用“␣␣␣#####␣␣”作为输出格式，其中的空格是为了使输出时各元素间隔开。又如，数组包含4列数据，第1列元素为4位无符号整数，第2列、第3列元素同为3位整数，第4列元素含3位整数，1位小数，此时，语句中可以选用如下格式串进行输出：

“␣␣␣######␣␣###␣␣␣###␣␣␣###.#”

其中的空格数可根据具体情况进行调整。

输出报表时，如果表格中的一行既有字符量，又有数值量，此时，一般不利用MAT PRINT　USING语句进行输出，而用双重循环结构，通过引用数组元素的方法完成制表操作。

〔**例8-3**〕　数据报表制作应用举例。

题目：设某班共有n名学生，本学期开设有：哲学、数学、英语、BASIC程序设计、体育等五门课程，要求编制程序输出全班所有学生各门课程成绩表。报表包括：

表名行

列标题行　姓名、各科名称、平均成绩。

个人成绩行　姓名、各科及平均成绩等。

合计行　全部学生各科的平均成绩，以及总平均成绩等。

分析：为了便于成绩统计和输出，将所有数据组织成三个数组。其中 name $ (50)数组用于存放学生姓名；score(50,6)数组用于存放全部学生的各科成绩以及每人的平均成绩；aver(6)数组用于存放所有学生各科的平均成绩以及总平均成绩。

受篇幅限制，DATA 语句中仅列出6个学生的数据。

算法如图8-1所示。

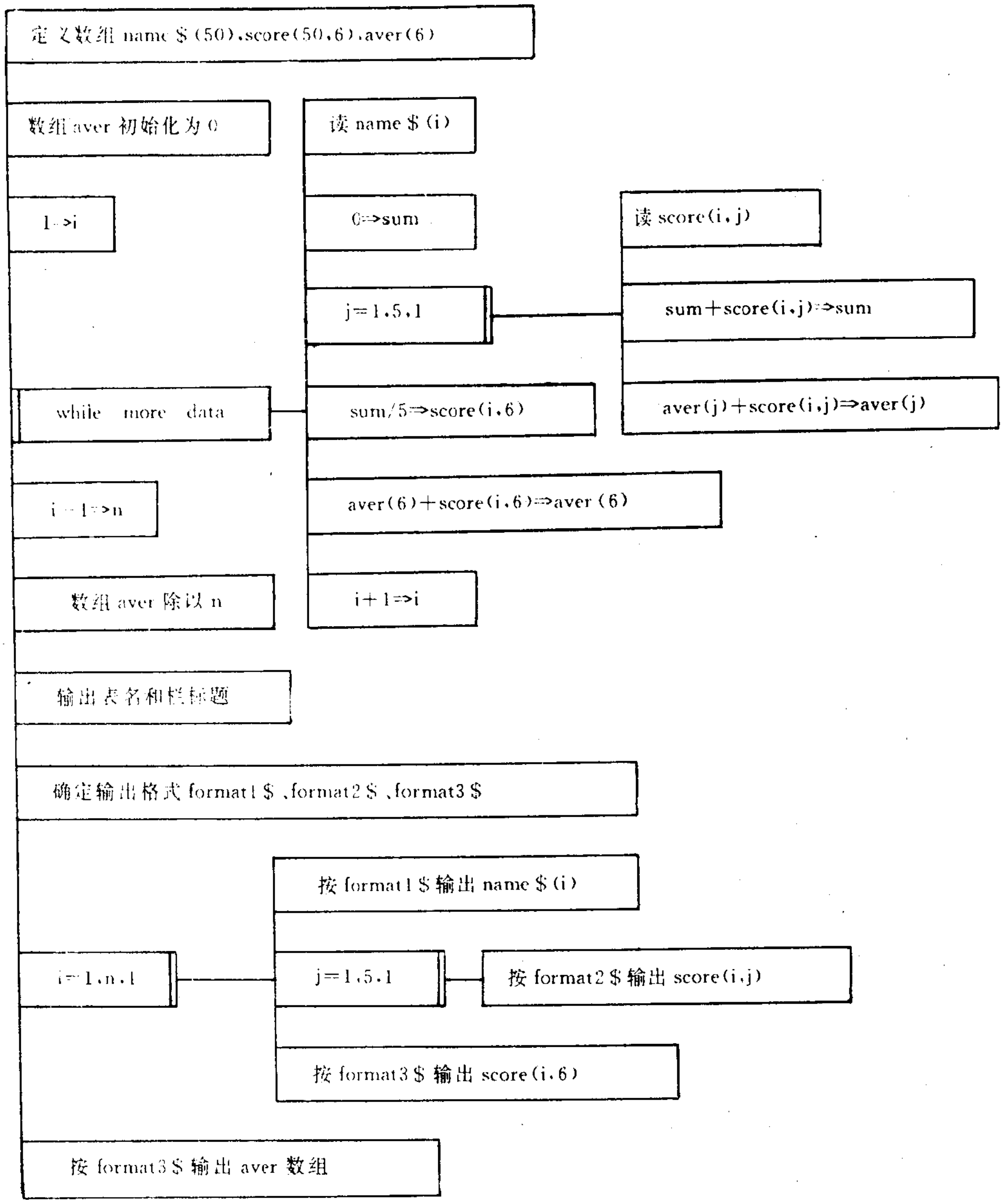

定义数组 name $ (50),score(50,6),aver(6)
数组 aver 初始化为 0
1=>i
while more data
读 name $ (i)
0=>sum
j=1,5,1
读 score(i,j)
sum+score(i,j)=>sum
aver(j)+score(i,j)=>aver(j)
sum/5=>score(i,6)
aver(6)+score(i,6)=>aver(6)
i+1=>i
i-1=>n
数组 aver 除以 n
输出表名和栏标题
确定输出格式 format1 $,format2 $,format3 $
i=1,n,1
按 format1 $ 输出 name $ (i)
j=1,5,1
按 format2 $ 输出 score(i,j)
按 format3 $ 输出 score(i,6)
按 format3 $ 输出 aver 数组

图8-1　成绩报表输出算法

程序为：

```
! Student score processing
DIM name $ (50),score(50,6),aver(6)

! Input data and clculate
MAT aver=0
LET i=1
DO while more data
    READ name $ (i)
    LET sum=0
    FOR j=1 to 5
        READ score(i,j)
        LET sum=sum+score(i,j)
        LET aver(j)=aver(j)+score(i,j)
    NEXT j
    LET score(i,6)=sum/5
    LET aver(6)=aver(6)+score(i,6)
    LET i=i+1
LOOP
LET n=i-1          ! n is number of students
MAT aver=(1/n) * aver

! Printing score list
PRINT tab(5,21);"Student score list"
PRINT repeat $ (" * ",58)
PRINT ":     name     :phil:     math:";
PRINT "engl:     basi:     spor:     aver:"
PRINT repeat $ (" * ",58)
LET format1 $ ="# # # # # # # # # # # # # # #"
LET format2 $ =" # # # #    "
LET format3 $ ="# # # #. # "
FOR i=1 to n
     PRINT using format1 $ :name $ (i);
     FOR j=1 to 5
          PRINT using format 2 $ :score(i,j);
     NEXT j
     PRINT using format3 $ :score(i,6)
     PRINT repeat $ ("-",58)
NEXT i
```

```
PRINT using format1 $ :"Total average";
MAT PRINT using format3 $ :aver
PRINT repeat $ ("-",58)
DATA Luo ping,91,81,79,84,86
DATA Fang yuan yuan, 89,84,92,94,82
DATA Jiao zhi ying, 70,55,43,65,78
DATA Ding zhen shan, 74, 71, 78, 82, 63
DATA Shu dong fang, 69,72,81,70,68
DATA Wang hong jun,82,94,79,90,80
END
```

执行 RUN 命令后，屏幕将转为正文显示模式，然后在指定位置显示如下数据报表。

```
                      Student score list
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
:    name          :  phil  :  math  :  engl  :  basi  :  spor  :  aver   :
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
   Luo ping           91       81       79       84       86       84.2
— — — — — — — — — — — — — — — — — — — — — — — — — — — — — — — — — — —
   Fang yuan yuan     89       84       92       94       82       88.2
— — — — — — — — — — — — — — — — — — — — — — — — — — — — — — — — — — —
   Jian zhi ying      70       55       43       65       78       62.2
— — — — — — — — — — — — — — — — — — — — — — — — — — — — — — — — — — —
   Ding zhen shan     74       71       78       82       63       73.6
— — — — — — — — — — — — — — — — — — — — — — — — — — — — — — — — — — —
   Shu dong fang      69       72       81       70       68       72.0
— — — — — — — — — — — — — — — — — — — — — — — — — — — — — — — — — — —
   Wang hong jun      82       94       79       90       80       85.0
— — — — — — — — — — — — — — — — — — — — — — — — — — — — — — — — — — —
   Total average      79.2     76.2     75.3     80.8     76.2     77.5

— — — — — — — — — — — — — — — — — — — — — — — — — — — — — — — — — — —
```

程序中共使用四个 PRINT USING 语句和一个 MAT PRINT USING 语句，请读者分析每一个语句的作用，并指出每个输出项使用的输出格式。

第二节　屏幕的定位输入

开发事务数据处理程序时，往往要求以填“票据”方式输入数据，即表框内分若干栏，输入数据时，各栏事先已显示出该栏目的名称，如“产品号”、“产品名”、“产地”等，并留有输入该栏数据的位置。输入数据时，光标首先被置于第一栏内，输入数据后，光标被控制移到第二栏，如

此反复，直到“票据”中的各栏数据输入完毕为止。

为了实现数据的屏幕定位输入，True BASIC 提供了一簇光标定位控制语句。

一、屏幕的光标定位控制

1. CLEAR(清屏)语句

执行 CLEAR 语句将清除屏幕上的当前内容，并把光标置于屏幕的左上角，即第一行、第一列位置。语句格式为：

```
CLEAR
```

执行 CLEAR 语句之后，显示器屏幕由初始状态（编辑窗口和背景窗口）转换为文本显示模式。此时，显示器的屏幕按坐标法，纵向分为25行，横向分为80列。

2. SET CURSOR(光标设置)语句

语句格式为：

```
SET  CURSOR 〈行数〉,〈列数〉
```

其中，行数和列数均为数值表达式，取值范围分别限制在1～25和1～80之间。执行该语句将把光标置于指定的坐标位置。

利用 SET CURSOR 语句将光标定位以后，执行 INPUT、PRINT 等输入输出语句时，所输入或显示数据就从指定位置开始显示。

〔**例8-4**〕 光标的定位输入和输出示例。

```
! Control position of cursor
CLEAR
SET cursor 5,10
PRINT "How are you?"
SET cursor 7,10
LINE INPUT reply $
SET sursor 9,10
PRINT reply $
PAUSE 30          !  pause 30 seconds
END
```

执行程序时，首先清屏幕，然后从第5行第10列开始显示字符串“How are you?”，自键盘输入的数据将从第7行第10列开始显示，第二个 PRINT 语句从第9行第10列开始显示变量 reply $ 的值。语句 PAUSE 为暂停语句，其作用是使程序运行暂停30秒钟，以便操作者观察屏幕显示信息。

3. ASK CURSOR(光标位置查询)语句

在重新确定光标位置之前，往往需要求光标的当前位置，此时可使用 ASK CURSOR 语句。其语句格式为：

```
ASK  CURSOR 〈行变量〉,〈列变量〉
```

格式中的行变量和列变量应该是数值型变量。执行该语句时，将把光标所在位置的行值和

列值赋给指定变量。

当用户在一个新的计算机硬件环境下编程时，可以利用 ASK MAX CURSOR 语句查询屏幕可以显示的最大行数和列数。其语句格式为：

```
ASK MAX CURSOR 〈行变量〉,〈列变量〉
```

4. 光标的打开与关闭

在正常状态下，光标处于打开状态。程序设计过程中，有时要求光标处于关闭状态。此时，可以用如下语句实现：

```
SET CURSOR OFF
```

执行该语句后，光标将从屏幕上消失。如果打算重新显示光标，可用如下语句实现：

```
SET CURSOR ON
```

True BASIC 还允许使用“ASK CURSOR〈字符串变量〉”语句查询光标处于开或者闭的状态。若光标处于打开状态，字符串变量将获得“ON”，否则，获得“OFF”。

二、GET KEY（单键输入）语句

GET KEY 语句的功能是，程序暂停运行，等待用户自键盘输入一个字符，输入后将该字符的 ASCII 码值赋给语句中的变量。语句格式为：

```
GET KEY 〈变量〉
```

与 INPUT 语句不同的是，GET KEY 语句将键入的键直接输入，不需要等待回车，并且该语句接收的是键入键的 ASCII 码值（十进制数），而不是键所代表的字符。程序设计中常利用该语句实现光标移动键（↑、↓、→、←）以及复合键（用 Shift、Ctrl、Alt 与其他键组合而成）的输入。

由于利用该语句输入的字符不在屏幕上显示，因此，程序中设置的操作口令，常利用该语句输入。因为口令是不应该被别人发现的。

GET KEY 语句的实质是从键盘缓冲区中获取一个字符。如果执行该语句时键盘缓冲区中已存有字符，那么系统就把该缓冲区中的第一个字符的 ASCII 码值传送给指定变量。这样，系统执行该语句时就不会暂停下来。

〔**例8-5**〕 数据的“票据”式输入示例。

分析：首先需要制作“票据”，为了简化设计，仅以图8-2所示的“票据”为例进行讨论。

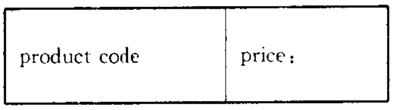


图8-2 简单票据示例

制“票据”时需要将制表符的 ASCII 码值用 chr $ 函数转换为相应的字符，以便显示。制表符“┌,─,┬,┐,│,┼,├,┤,└,┴,┘”的 ASCII 码分别为218,196,194,191,179,197,195,180,192,193和217。设图8-2所示的票据其左上角为屏幕第10行第20列，并且产品代码和单价数据的长度分别为6和7。

程序为：

```
! Input of formating
LET row = 10
LET col = 20
CALL draw_frame(row, col)
PRINT tab(row+1,col+2);“product code:”;
PRINT tab(45);“price:”
PRINT tab(row+1,36);reprea $ (chr $ (219),6)
CALL get_item(row+1,36,6,product_code $ )
PRINT tab(52);repeat $ (chr $ (219),7)
CALL get_item(row+1,52,7,p $ )
LET price=val(p $ )
PRINT tab(20,20);“product-code is”;product_code $
PRINT tab(22,20);“price is”;price
END

! ------------------------------------------------------------
SUB draw_frame(row,col)
      PRINT tab(row,col);chr $ (218);
      PRINT repeat $ (chr $ (196),22);chr $ (194);
      PRINT repeat $ (chr $ (196),16);chr $ (191)
      PRINT tab(row+1,col);chr $ (179);
      PRINT tab(43);chr $ (179);
      PRINT tab(60);chr $ (179)
      PRINT tab(row+2,col);chr $ (192);
      PRINT repeat $ (chr $ (196),22);chr $ (193);
      PRINT repeat $ (chr $ (196),16);chr $ (217)
END SUB

! ------------------------------------------------------------
SUB get_item(line,col,len, s $ )
      LET s $ =“”
      FOR i=0 to len-1
            GET KEY a
            PRINT tab(line,col+i);chr $ (a);
            LET s $ =s $ &chr $ (a)
      NEXT i
END SUB
! ------------------------------------------------------------
```

程序中，子程序 draw_frame 用来绘票据的框架，其中参数 row，col 为框架的左上角屏幕坐标值。子程序 get_item 用于在指定位置输入规定长度(len)的数据，参数 s＄将返回输入的数据。主程序中的 PRINT 语句使用了标准函数 repeat＄(chr＄(219)，6)，其作用是从光标当前位置显示一长度为6的光条(带)。

三、KEY INPUT(输入测试)函数

程序设计中，有时为方便用户，允许按任意键作为程序是否继续执行的条件。True BASIC 提供的 KEY INPUT 函数可以测试用户是否按下一个键。函数的格式为：

```
KEY  INPUT
```

如果按下任意一个键，函数值为真(True)，否则为假(false)。

〔**例8-6**〕 KEY INPUT 函数应用示例

```
   ⋮
PRINT "Press any key to continue"
DO
LOOP until key input
   ⋮
```

程序执行完 PRINT 语句后，将不断地执行 DO 循环，直到按下任意一键，循环才会终止，并继续执行后续语句。这一结构往往用于检查一段程序的运行状态是否正常，如打印机是否准备好，磁盘是否插入驱动器等，当这些准备工作做好后，按下任意键后程序将继续执行。

需要补充说明的是，如果这一结构之后有一个 GET KEY 语句，那么在执行该语句时系统并不会暂停，其原因是它已接收执行 DO 循环时所敲入的字符。为了使执行 GET KEY 语句时暂停，可在 DO 循环与该 GET KEY 语句之间插入一个新的 GET KEY 语句，以便把键盘缓冲区中的字符送入一个辅助变量中。

第三节　菜单驱动程序设计

一个规模较大的程序，往往有多种功能供用户选择。然而，就用户而言，要记住程序所具有的所有功能绝非易事。因此，为了提高效率，简化操作，一般采用人机交互方式，即常说的菜单驱动方式。运行程序时首先在屏幕上显示出标识程序功能的文字说明或图形(即菜单)，操作者只要通过某种简单的方式(比如使用光标键或鼠标器，或得选择1，2，3等数字，或者 A，B，C 等字母)选定其中的某项功能(即选菜单)，程序便会根据操作者的选择，由选择结构程序段去选择执行相应的功能模块。菜单驱动方式不仅使操作者感到轻松自如，操作简便灵活，而且可以减少差错，因而近年来得到越来越广泛的应用。

程序实现菜单驱动的前提是按模块化方法设计程序。一个规模较大的数据处理系统往往由一个主控模块和若干个功能模块组成，而功能模块又可细化并由多个子模块组成。实现主控模块→ 功能模块→功能子模块的调用往往需要采用多级菜单驱动。

菜单的选取方式可以分为两类：根据功能名称或数字选取菜单和光标控制选取菜单。后一类又可细分为横移式和下拉式选取菜单等。

一、根据功能名称或数字选取菜单

设计菜单驱动程序，首先要设计菜单的画面。按照功能名称或数字选取菜单时，菜单画面的设计通常是联合使用 CLEAR、SET CURSOR、PRINT 语句，在屏幕的指定位置列出模块功能清单，以及选取各功能模块的数字或名称（首字母）。例如，某个数据处理系统的主控模块包括四个功能，即：输入数据；统计；检索；退出。其画面设计如下：

```
CLEAR
SET cursor 10,1
PRINT tab(8);"* * * min menu * * *"
PRINT "————————————————————————"
PRINT ":       input data···················· 1:"
PRINT ":       statistics···················· 2:"
PRINT ":       search························ 3:"
PRINT ":       end··························· 4:"
PRINT "————————————————————————"
```

为了实现菜单的功能选择，程序必须将用户选择的结果赋给某一变量，使分支程序根据变量的值来控制程序流程以完成相应的功能。一般地，接收选择代码的方式用 INPUT 语句实现。如：

```
INPUT prompt "Please enter your chose:":key_m
```

如有特殊要求，也可以用 GET　KEY 语句接收选择代码。

实现分支选择的方法，可以选用 SELECT　CASE 语句，也可以选用 IF-THEN-ELSE END IF 语句。如

```
SELECT CASE key_m
CASE 1
      CALL in_data
CASE 2
      CALL statistics
CASE 3
      CALL search
CASE else
      PRINT "The end!"
END SELECT
```

菜单画面设计和功能代码接收，通常设置为一个逻辑模块，实现分支选择应设置为另一逻辑模块，并且应提供容错功能。即对于操作者可能产生的误操作，应允许重新选择，而不应当使程序“死锁”或退回到 True BASIC 状态。

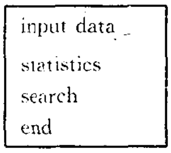
二、下拉式菜单驱动程序的设计

对于一个由多级菜单驱动的应用程序，通常要设计几个菜单模块，为了提高编程效率，下面介绍通用的下拉式菜单驱动程序的设计。

1. 何谓下拉式菜单驱动程序

下拉式菜单驱动程序也是一种菜单驱动程序，与上述菜单驱动程序的不同之处在于选项的方式不同，即在屏幕上的一个矩形框各行内列出本级菜单各功能项的文字信息（如功能名称），并以光标或光条的形式表明当前所指功能，用户选项时只需利用光标上下移动键（↑和↓）将光标或光条向上或向下移动，选中某项功能后，按下某一功能键或者回车键后，系统便转向指定功能的处理，结束之后将重新返回该菜单，或者结束应用程序的运行。

2. 下拉式菜单子程序的设计



利用光标上下移动键（↑和↓）选项，用回车键表示选中此项。

图8-3　下拉式菜单选取示意图

为了便于讨论，图8-3给出了仅有四个功能项的某级菜单画面。显示菜单画面和选取菜单子程序的算法可设计为：

```
s1：利用制表符显示菜单的边框。
s2：显示菜单各功能项的文字信息。
s3：循环处理：
  s3.1：将光标指向第 i 项。
  s3.2：自键盘接收一个键，然后判断，若是
    ↓键：i 加1，置选取项为－1，表示越过此项；
    ↑键：i 减1，置选取项为－1，表示越过此项；
    回车键：选中此项，置选取项为 i，并退出循环；
    ESC 键：不选任何项，置选取项为0，并退出循环。
```

子程序参数说明：

row，col：菜单左上角在屏幕上的行、列位置。

n，l：菜单中的项目个数及菜单宽度。

m $ ()：为数组参数，其各元素值为菜单各项目的名称。

choice：选取项。

子程序为：

```
! ————————————“Menulib. TRU”————————————
EXTERNAL
SUB menu(row,col,n,l,m$(),choice)
    CALL draw_frame(row,col,n,l)
    FOR i=1 to n
        PRINT tab(row+i,col+1);
        PRINT m$(i)&repeat$(“  ”,l-len(m$(i)))
    NEXT i
    ! Choicing item
    LET i=max(1,choice)
    DO
```

```
          GET CURSOR row+i,col+l+2
          LET i0=i
          CALL pick(n,i,choice)
          IF choice>=0 then EXIT SUB
          LOOP
END SUB

! ------------------------------------------------------------------
! Drawing frame
SUB draw_frame(row,col,n,l)
     PRINT tab(row,col);chr $ (218);
     PRINT repeat $ (chr $ (196),l+2);chr $ (191)
     FOR i=1 to n
          PRINT tab(row+i,col);chr $ (179);
          PRINT repeat $ ("    ",l+2);chr $ (179)
     NEXT i
     PRINT tab(row+n+1,col);chr $ (192)
     PRINT repeat $ (chr $ (196),l+2);chr $ (217)
END SUB

! ------------------------------------------------------------------
! Choicing menu
SUB pick(n,i,choice)
     LET choice=-1
     GET KEY code
     SELECT CASE code
     CASE 328
          LET i=i-1
     CASE 336
          LET i=i+1
     CASE 13
          LET choice=i
     CASE 27
          LET choice=0
     CASE else
          LET choice=choice
     END SELECT
     LET i=mod(i-1,n)+1
END SUB
```

```
! ------------------------------------------------------------
```

说明:“↑”、“↓”、回车键、ESC键的ASCII码值分别为328、336、13、27。语句“LET i=mod(i-1,n)+1”的作用是使i在1到n的范围内循环。另外,为了方便其他程序调用,在子程序menu之前增加了EXTERNAL语句,并在调试完毕后以“menulib.TRU”为文件名存入磁盘,以作为一个库文件。

3. 下拉式菜单子程序的调用

下面以图8-3列出的菜单选择项目为例,给出菜单子程序的调用程序。

```
LIBRARY "menulib.TRU"
DECLARE DEF max_length
DIM mnu$(4)
MAT READ mnu$
LET lenth=max_length(mnu$())
DO
   CALL menu(10,30,4,length, mnu$(),choice)
   CLEAR
   SELECT CASE choice
   CASE 1
         CALL input_data(…)
   CASE 2
         CALL statistics(…)
   CASE 3
         CALL search(…)
   CASE 0,4
         EXIT DO
   END SELECT
   CLEAR
LOOP
PRINT "* * * * * * * * * * * * * * * * *"
PRINT "*                bye-bye         *"
PRINT "* * * * * * * * * * * * * * * * *"
DATA "input data","statistics","search","end"
END

! ------------------------------------------------------------
DEF max_length(s$())
    LET l=lbound(s$)
    LET u=ubound(s$)
    LET max_l=len(s$(l))
    FOR i=l+1 to u
```

```
        LET max_l=max(max_l,len(s$(i)))
    NEXT i
    LET max_length=max_l
END DEF
! ----------------------------------------------------------------
```

上述程序中，外部函数 max_length 实现计算字符串型数组各元素其长度的最大值，并以此值确定菜单画面的宽度。受篇幅限制，主程序中仅给出与实现菜单驱动程序有关的程序段和语句，省略了菜单各功能项目的处理过程。

4. 菜单当前状态的存储与再现

多级菜单驱动程序时，下一级菜单将覆盖当前菜单，因此，需要将当前菜单的状态存储起来，以利于下一级菜单返回时再现。下面给出实现菜单当前状态的存储和再现功能的子程序 keepmenu(row,col,n,l,k$)和 showmenu(k$)，其中，前者用来实现菜单当前状态的存储，参数 k$ 用于接收左上角为(row,col)的 n+2行，l+4列屏幕内容。后者用于实现将存入 k$ 中的内容在屏幕上再现。

子程序如下：

```
! Keeping menu
SUB keepmenu(row,col,n,l,k$)
    LET k$=chr$(row)&chr$(col)&chr$(n)&chr$(l)
    FOR i=row to row+n+1
        FOR j=col to col+3+l
            LET address=7536663+(i-1)*160+2*j-1
            LET k1$=chr$(peek(address))
            LET k2$=chr$(peek(address+1))
            LET k$=k$ & k1$ & k2$
        NEXT j
    NEXT i
END SUB

! ----------------------------------------------------------------
! Showing menu
SUB showmenu(k$)
    LET row=ord(k$[1:1])
    LET col=ord(k$[2:2])
    LET n=ord(k$[3:3])
    LET l=ord(k$[4:4])
    LET t=4
    FOR i=row to row+n+1
        FOR j=col to col+l+3
            LET t=t+2
            LET address=753663+(i-1)*160+2*j-1
            CALL poke(address,ord(k$[t-1:t-1]))
            CALL poke(address,ord(k$[t:t]))
        NEXT j
    NEXT i
END SUB
! ---------------------------------------------------
```

在子程序 keepmenu 之前增加 EXTERNAL 语句，然后利用 SAVE 命令将上述子程序以“kpshwlib. TRU”为文件名存入磁盘，作为库文件，以便于其他程序调用。

需要指出的是，上述子程序涉及标准函数 peek，标准子程序 poke 以及屏幕存储单元的地址等内容，读者开发实用程序时，可直接调用，而不必细究。

第四节　菜单驱动程序设计举例

按模块化程序设计方法编写程序，实现某系组织的英语、数学、计算机三门课程的竞赛成绩管理。完成如下功能：

在人机交互方式下输入全部原始数据，并提供数据校验功能。

查询处理：按学号查询；按姓名查询。

统计处理：全部参赛成绩列表输出；超过平均分数者列表输出；不及格者列表输出。

遵循模块划分原则，程序结构如图8-4所示。

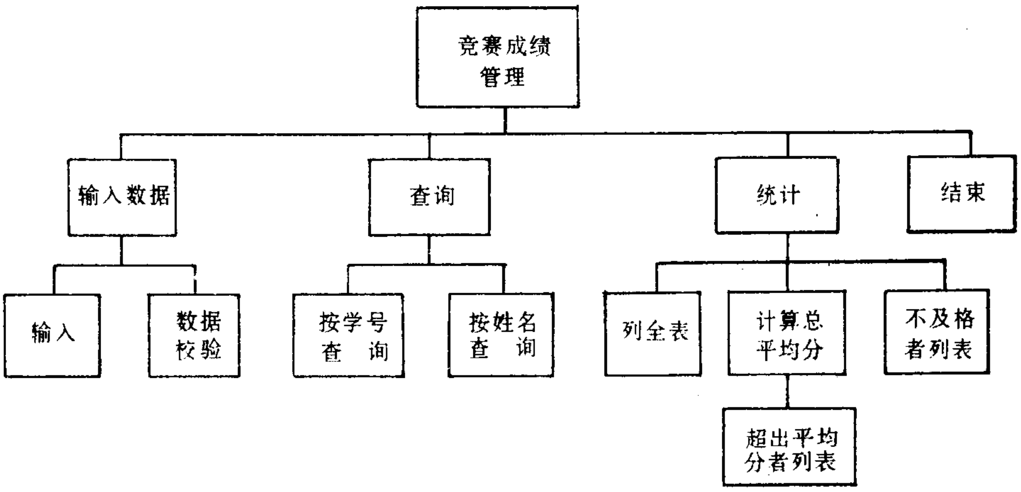

竞赛成绩管理
输入数据
查询
统计
结束
输入
数据校验
按学号查询
按姓名查询
列全表
计算总平均分
不及格者列表
超出平均分者列表

图8-4　竞赛成绩管理程序结构图

程序为：

```
! * * * * * * * * * * * * * * * * * * * * * * *
! * contest score manaement                     *
! * date written: 1994.1.15.                   *
! * computer: AST386                            *
! * * * * * * * * * * * * * * * * * * * * * * *
! Main program
DIM st_no(1),name$(1),score(1,4)
DO
    CALL main_menu(key_m)
    SELECT CASE key_m
    CASE 1
         CALL in_data(st_no(),name$(),score(,))
    CASE 2
         CALL statistics(st_no(),name$(),score(,))
    CASE 3
         CALL search(st_no(),name$(),score(,))
    CASE ELSE
         PRINT "End !"
    END SELECT
LOOP until key_m = 4
CLEAR
SET CURSOR 10,1
PRINT "* * * * * * * * * * * * * * * * * * * * * *"
PRINT "*              Good bye!                   *"
PRINT "* * * * * * * * * * * * * * * * * * * * * *"
PAUSE 10
END

! ------------------------------------------------------------
! Display main menu
SUB main_menu(key_m)
    DO
        CLEAR
        SET CURSOR 10,1
        PRINT tab(8);"main menu"
        PRINT "* * * * * * * * * * * * * * * * * * * * * *"
        PRINT "* input data••••••••••••••••••••••••••••••1 *"
        PRINT "* statistics•••••••••••••••••••••••••••••• 2 *"
        PRINT "* search•••••••••••••••••••••••••••••••• 3 *"
        PRINT "* end •••••••••••••••••••••••••••••••••• 4 *"
        PRINT "* * * * * * * * * * * * * * * * * * * * * *"
        PRINT
```

```
          PRINT "Enter your selection(1/2/3/4):";
          INPUT key_m
          IF key_m<1 or key_m>4 then
             PRINT "Code entered error, try again!"
             PAUSE 5
          ELSE
              EXIT DO
          END IF
     LOOP
END SUB

! ------------------------------------------------------------------
! Input data modules
SUB in_data(st_no(),name $ (),score(,))
     INPUT prompt "How many students:":n
     MAT redim st_no(n),name $ (n),score(n,4)
     FOR i=1 to n
          CALL inandcheck(no, nam $ ,eng,mat,com,ave)
          LET st_no(i)=no
          LET name $ (i)=nam $
          LET score(i,1)=eng
          LET score(i,2)=mat
          LET score(i,3)=com
          LET score(i,4)=ave
     NEXT i
END SUB

! -------------------------------------------------------
! Input and check data
SUB inandcheck(no,name $ ,eng,mat,com,ave)
     LET replay $ =""
     DO
          CLEAR
          SET CURSOR 10,1
          INPUT prompt "stdent-no————→":no
          INPUT prompt "name  —————→":nam $
          INPUT prompt "english—————→":eng
          INPUT prompt "mathemetics  —→":mat
          INPUT prompt "computer————→":com
```

```
        PRINT
        LET sum=eng+mat+com
        LET ave=round(sum/3,1)
        CALL dis_title
        CALL dis_score(no,nam $ ,eng,mat,com)
        PRINT "Is the data correct?(Y/n)"
        INPUT reply $
        IF ucase $ (reply $ )="Y" then
          EXIT SUB
        ELSE
            PRINT "Data entered error, try again!"
            PAUSE 5
        END IF
    LOOP
END SUB

! ------------------------------------------------
! Display title of report
SUB dis_title
    PRINT repeat $ ("=",61)
    PRINT ":      stu-no      :         name            :";
    PRINT "          engl   :   math    :   comp       :"
    PRINT repeat $ ("=",61)
END SUB

! Display a student score
SUB dis_score(no,nam $ ,score1,score2,score3)
    PRINT using "# # # # # # # #      :";no;
    PRINT using "# # # # # # # # # # # # # # # # # # # :";nam $ ;
    PRINT using "# # # # #      :";score1,score2,score3
    PRINT repeat $ ("-",61)
END SUB

! ------------------------------------------------
! Search modules
SUB search(no(),nam $ (),s(,))
    DO
        CALL search_menu(key_s)
        SELECT CASE key_s
        CASE 1
              CALL search_stno(no(),nam$(),s(,))
        CASE 2
              CALL search_name(no(),nam$(),s(,))
        CASE ELSE
              PRINT "Return !"
        END SELECT
   LOOP until key_s = 3
END SUB

! ------------------------------------------------
! Display search menu
SUB search_menu(key_s)
   DO
        CLEAR
        SET CURSOR 10,1
        PRINT tab(11);"search menu"
        PRINT "* * * * * * * * * * * * * * * * * * * * * * * * * *"
        PRINT " *  search by st-no································1  * "
        PRINT " *  search by name································2  * "
        PRINT " *  exit··········································3  * "
        PRINT "* * * * * * * * * * * * * * * * * * * * * * * * * *"
        PRINT
        PRINT "Enter your selection (1/2/3):";
        INPUT key_s
        IF key_s < 1 or key_s > 3 then
           PRINT "Code entered error,try again!"
           PAUSE 5
        ELSE
             EXIT DO
        END IF
   LOOP
END SUB

! ------------------------------------------------------------
! Search by student-no
SUB search_stno(no(),nam$(),s(,))
```

```
    DO
        CLEAR
        LET flag=0
        PRINT
        PRINT “Enter st-no to find:”;
        INPUT st_no_f
        INPUT l=lbound(no)
        LET stunum=ubound(no)
        PRINT
        FOR i=l to stunum
            IF no(i) = st_no_f then
               LET p=i
               LET flag=1
               EXIT FOR
            END IF
        NEXT i
        IF flag <> 1 then
           PRINT st_no_f;“not to be found!”
        ELSE
            PRINT
            LET n $ =nam $ (p)
            LET s1=s(p,1)
            LET s2=s(p,2)
            LET s3=s(p,3)
            CALL dis_title
            CALL dis_score(no(p),n $ ,s1,s2,s3)
            PRINT
        END IF
        INPUT prompt “Continue? (Y/N)”:ans $
    LOOP until ucase $ (ans $ )=“N”
END SUB

! -----------------------------------

! Search by name
SUB search_name(no(),nam $ (),s(,))
    DO
        CLEAR
        LET flag=0
        PRINT
```

```
          PRINT "Enter the name to find:";
          INPUT s_name $
          PRINT
          LET l=lbound(no)
          LET stunum=ubound(no)
          FOR j=l to stunum
               LET n $ =nam $ (j)
               IF lcase $ (n $ )=lcase $ (s_name $ ) then
                  LET flag=1
                  LET s1=s(j,1)
                  LET s2=s(j,2)
                  LET s3=s(j,3)
                  CALL dis_title
                  CALL dis_score(no(j),n $ ,s1,s2,s3)
                  PRINT
               END IF
          NEXT j
          IF flag = 0 then
                PRINT st_name $ ;" not tot be found!"
                PRINT
          END IF
          INPUT prompt "Continue? (Y/N)":ans $
     LOOP UNTIL ucase $ (ans $ )="N"
END SUB

! ----------------------------------------
! Statistics modules
SUB statistics(no(),nam $ (),s(,))
     DO
          CLEAR
          CALL stat_menu(key_st)
          SELECT CASE key_st
          CASE 1
                CALL r_all_stdnt(no(),nam $ (),s(,))
          CASE 2
                CALL r_highr_aver(no(),nam $ (),s(,))
          CASE 3
                CALL r_failed(no(),nam $ (),s(,))
```

```
        CASE else
             PRINT "Return !"
        END SELECT
    LOOP until key_st = 4
END SUB .

! ------------------------------------------------------------------
! DISPLAY statistics menu
SUB stat_menu(key_st)
    DO
        CLEAR
        SET CURSOR 10,1
        PRINT tab(8);"statistics menu"
        PRINT "************************"
        PRINT "* report of all students························1 *"
        PRINT "* report of higher than average ················2 *"
        PRINT "* report of those failed························3 *"
        PRINT "* exit··········································4 *"
        PRINT "************************"
        PRINT
        PRINT "Enter your selection(1/2/3/4):";
        INPUT key_st
        IF key_st <1 or key_st >4 then
           PRINT "Code entered error, try again!"
           PAUSE 5
        ELSE
            EXIT DO
        END IF
    LOOP
END SUB

! ------------------------------------------------------------------
! Display report of all students
SUB r_all_stdnt(no(),nam$(),s(,))
    CLEAR
    SET CURSOR 10,1
    LET l=lbound(no)
    LET stunum=ubound(no)
    PRINT tab(25);"Score Report"
    CALL dis_title
    FOR j=1 to stunum
        LET n $ =nam $ (j)
        LET s1=s(j,1)
        LET s2=s(j,2)
        LET s3=s(j,3)
        CALL dis_score(no(j),n $ ,s1,s2,s3)
    NEXT j
    PRINT
    PAUSE 30
END SUB

! ------------------------------------------------------------------
! Display report of higher than average
SUB r_highr_aver(no(),name $ (),s(,))
    LET l=lbound(no)
    LET stunum=ubound(no)
    LET total=0
    FOR j=1 to stunum
        LET total=total+s(j,4)
    NEXT j
    LET aver=round(total/stunum,1)
    CLEAR
    SET CURSOR 10,1
    PRINT tab(15);
    PRINT “Score Report Of Higher Than aver”
    PRINT repeat $ (“   ”,40);
    PRINT “ * * *  average=”;aver;“ * * * ”
    CALL dis_title
    FOR j=1 to stunum
            IF S(j,4)>=aver then
            LET n $ =nam $ (j)
            LET s1=s(j,1)
            LET s2=s(j,2)
            LET s3=s(j,3)
            CALL dis_score(no(j),n $ ,s1,s2,s3)
        END IF
    NEXT j
    PRINT
    PAUSE 30
END SUB

! ----------------------------------------------------------
! Display report of those failed
SUB r_failed(no(),nam$(),s(,))
    CLEAR
    SET CURSOR 10,1
    LET l=lbound(no)
    LET stunum=ubound(no)
    PRINT tab(20);“Score Failed Report”
    CALL dis_title
    FOR i=l to stunum
        FOR j=1 to 3
            IF s(i,j)<60 then
               LET s1=s(i,1)
               LET s2=s(i,2)
               LET s3=s(i,3)
               LET n$=nam$(i)
               CALL dis_score(no(i),n$,s1,s2,s3)
               EXIT FOR
            END IF
        NEXT j
    NEXT i
    PAUSE 30
END SUB
! ----------------------------------------------------------
```

上述程序由主程序和若干个外部子程序组成。通过主菜单提示，由用户选择输入功能项号，以转到相应的功能模块进行处理，在检索和统计模块中，又给出二级菜单，以便实现不同目的的检索和统计处理。

在原始数据和菜单选择项号的输入中，程序模块中采用了校验和复式输入法，即对错误的输入，程序并不接受，并提供重新输入的功能。这一技术在开发实用程序时是经常用到的。

在按学号检索的模块中，采用了顺序检索算法。读者可修改该模块，实现对分检索算法，以提高检索效率。

分析上述程序，读者还会发现，输入数据、检索和统计等模块与主程序之间均需传递三个数组。有兴趣的读者可以将上述相关模块定义为 MODULE-END MODULE 结构，并把这三个数组定义为公用数组，这样可以减少形式参数的个数，以及逻辑模块之间的数据传递。

根据上节讲述的通用下拉式菜单以及当前菜单状态的存储和再现，读者可将上述程序中的菜单设计及选择修改为下拉式菜单驱动程序的形式。

习　题

8—1. 利用 TAB 函数，输出正弦曲线。

8—2. 在同一屏幕上，用不同字符输出下面两条函数曲线，要求每隔10°显示一个字符。

$$\begin{cases} y1 = \sin x & (-\pi < x \leqslant \pi) \\ y2 = \dfrac{1}{3}x & (-\pi < x \leqslant \pi) \end{cases}$$

8—3. 试说明 INPUT 语句与 GET KEY 语句之间的异同。

8—4. 判断下述用于数值型数据输出的格式串是否合法？为什么？

(1)“##%%.%%”　　(2)“+##.##”

(3)“>######”　　(4)“$$,$##.#”

(5)“$***.**”　　(6)“###.$$”

8—5. 设 a=125.824，b=−1029.63，c=14.723，若在 PRINT USING 语句中用下列格式串输出它们的值，将得到什么结果？

(1)“######”:a,b,c

(2)“%%%%.%%”:a,b,c

(3)“####.##　****.*　%.%%”:a,b,c

(4)“++++.##　----.##　$$$#.##”:a,b,c

(5)“#.##∧∧∧∧　##.#∧∧∧∧”:a,b,c

8—6. PRINT USING 语句中使用下列格式串输出字符串，将得到什么结果？

(1)“<**,***,*##.##”:“True␣BASIC”

(2)“>$$$$$$$$$#.##”:“Programming”

8—7. 试编制程序，屏幕显示〔0,100〕之间各整数的平方根表，结果应精确到第3位小数。

8—8. 刘峰要打印一份校友名录表，表格栏目包括：姓名、性别、年龄、职务、毕业日期、现工作单位、邮政编码、电话号码、通讯处。请你帮助他设计程序。

8—9. 设职工工资原始数据包括：职工号、姓名、基本工资、洗理费、奖金、房租、水电费、病事假费、其他1、其他2。试编写程序，输出某生产车间工资清单。输出项包括：职工号、姓名、基本工资、洗理费、奖金、应发工资、房租、水电费、病事假费、其他1、其他2、应扣款额、实发工资。

其中，应发工资=基本工资+洗理费+奖金+其他1

应扣款额=房租+水电费+病事假费+其他2

实发工资=应发工资−应扣款额。

8—10. 东风制药厂欲建立人事档案管理系统，每个职工包括以下数据：

职工号、姓名、性别、出生年月、籍贯、学历、职务、参加工作日期。

试利用菜单驱动技术编程实现：①分别按职工号、姓名、年龄、学历、参加工作日期对职工进行查询；②除按职工号、姓名查询以外，其他查询均需报表输出；③采用分页输出技术，实现全体职工报表输出。

第九章　磁盘数据文件

随着计算机科学技术的发展，计算机不仅用于科学计算，而且更广泛地用于机关事务处理，企业管理，图书资料检索等非数值计算领域。在这些领域中，为实现各种目的所编的应用程序一般篇幅较长，需要处理的数据量较大，而且这些程序和数据常常需要保存起来，以便多次使用。然而计算机的内存容量有限，且一旦断电，内存中的信息就会消失。所以需要长期保留的程序和数据不能存于内存中。另外，如果大量的数据以 DATA 语句依附于程序中，即程序与数据不分离，那既限制了程序的通用性，又给数据的维护与修改带来困难。由于外存（对于微机，一般为磁盘）容量大，可不受内存空间的限制，且能长期保留存储在其上面的信息，因此，在当今计算机应用中，往往把需要多次使用的程序和数据分离，并分别以程序文件和数据文件的形式存储在外存上。当要使用这些程序和数据时，再把它们从外存调入到内存中来。

程序文件保存的是程序，数据文件保存的是数据。在 True BASIC 中，对程序文件的存取、连接、删除等操作，通过执行有关简单命令即可实现；对数据文件中数据的存取、检索、运算、更新等操作，必须通过运行一个程序，执行特定的语句才能实现。

在 True BASIC 中，数据文件按其存取方式与组织形式不同分为正文文件，记录文件和字节文件三种。本章将主要介绍这三种数据文件的有关概念及操作管理方法。

第一节　文件基本操作

一、通道与文件指针

把内存中的数据记到磁盘上的数据文件中，对内存称为输出，对磁盘称为写；反过来，把磁盘数据文件中的数据调入内存，对内存称为输入，对磁盘称为读。True BASIC 对一个数据文件进行输入输出操作之前，必须为该文件分配一个通道。这里所说的通道是软件的逻辑通道，它指出了数据在内存与磁盘之间传送的途径。通道的确定是通过 OPEN（打开文件）语句实现的。通道确定后，对文件的操作就变成了对相应通道的操作，在有关文件操作语句中，就不写文件名，而用相应的通道来代表该文件。

True BASIC 中的通道用通道说明符“#”后跟一个整数 n 来表示。n 的取值范围在1～999之间，不为整数时，系统自动四舍五入取整。屏幕和键盘的专用通道号为0，此通道总是打开的，不能关闭。

与 READ/DATA 语句执行时的数据指针相类似，数据文件的输入输出过程中也有一个“文件指针”，这个概念用来表示文件读写到什么地方，指针总是指向下一步要操作的数据，其位置随着文件读写操作的过程而发生变化。

二、OPEN（打开文件）语句

在建立或使用一个数据文件之前，必须用 OPEN 语句打开该文件。其语句格式为：

```
OPEN #n: name〈文件标识〉
 ┌          ┌input ┐       ┌new   ┐ ┐
 │,access<  output >,creat< old    > │
 │          └outin ┘       └newold┘ │
 │               ┌text  ┐            │
 │,organizaton<  record >,recsize〈记录长度〉
 └               └byte  ┘            ┘
```

OPEN 语句的作用是为指定文件分配一个通道，建立起文件指针，并使它指向文件开始处。

语句中的 n 为通道号，是一个1～999之间的任意整数。name 后的文件标识为“盘符：文件名·扩展名”，若为当前盘，“盘符：”可省略，文件名及扩展名应符合操作系统对磁盘文件的规定，省略扩展名时，True BASIC 系统自动加上扩展名“TRU”。方括号中的各项为任选项，其中 access 用来指定正文文件和字节文件的存取方式，create 用来指定文件的建立模式，organization 用来指定文件的组织形式，即文件类型（在 True BASIC 2.03版中，organization 可简写为 org），recsize 用于指定字节文件中一次读取的字节数或新建立的记录文件的记录长度。各任选项之间用逗号分隔，它们出现的次序是任意的，只要在 name 文件标识项之后即可。如果选择了某任选项，其后花括号中的内容必选其一，各选择项的内容及意义如表9-1所示。

表9-1

选择项	选择内容	意义	备注
access	input output outin	只能读出 只能写入 读或写（可省略）	适用于正文文件和字节文件
creat	new old newold	建立新文件 使用已存在的文件（可省略） 其文件存在则使用，否则就建立新文件	
organization	text record byte	正文文件 记录文件 字节文件	
recsize	字符串长度 记录长度	指明一次读取的字节数 指明记录长度	适用于读字节文件 适用新建立的记录文件

例如：

OPEN #1:“B:EX”,access output, creat new, org text

表示在 B 盘上将建立一个新的正文文件，文件名为 EX，为该文件分配的通道号是1，准备对文件写数据。

OPEN #2:name “DATA”,create new, org record, recisize 64

表示在当前盘上建立名为 DATA 的一个新记录文件，其记录长度为64字节，分配给该文件的通道号是2，记录文件不指定访问模式。

OPEN #5:name “FILE”,access input, org byte

表示在当前盘上打开一个已经存在的字节文件“FILE”，分配的通道号为5，准备从文件中读取数据。

需要指出，一个 OPEN 语句，只能打开一个文件，同时打开多个文件应使用多个打开语句，且用不同的文件名和通道号加以区分。True BASIC 允许在一个程序中至多可同时打开10个文件。

三、CLOSE（关闭文件）语句

对一个数据文件进行输入输出操作结束之后，应及时将该文件关闭。关闭文件是通过 CLOSE 语句实现的，其语句格式为：

```
CLOSE #n
```

其中 n 为通道号，取值与相应 OPEN 语句中的相同。

该语句的作用是将通道号 n 对应的文件关闭，即切断该通道与文件的联系，释放出文件所占据的通道。

例如：

CLOSE #5

表示关闭与通道号5相联系的那个文件。

当程序执行到 END 语句时，如果在此之前未执行 CLOSE 语句，系统也将把所有打开的文件自动关闭。但我们仍建议读者，适时地书写 CLOSE 语句，这样可避免因某种事故造成文件中数据的丢失，同时也保证文件操作过程的完整性。

需要说明的是，在外部函数或外部子程序中也可以打开文件通道，然而这种通道是局部通道，即作用范围仅限于该函数和子程序中。当从外部函数或外部子程序返回调用程序时，无论是否执行 CLOSE 语句，系统将关闭局部通道。另外，通道也可作为子程序的形式参数，如#999。调用子程序时，将换成实际的通道号。

四、文件逻辑函数与设置文件指针语句

用 OPEN 语句打开文件时，文件指针总是指向文件的开始处，以后每读写一次，指针就自动后移一个单位，指向下一次要读写的数据。这个单位对于组织形式不同的文件，其含义是不同的，对正文文件是一个数据行，对记录文件是一个记录，对字节文件是一次读写的字节数。每个已建立的文件末尾都有一个结束标志，称为尾标志。当指针指到尾标志时，就不能对文件进行读操作了，否则将出错。为判断文件指针是否达到了尾标志，True BASIC 提供了以下两个逻辑函数。

MORE #n

END #n

如果通道号 n 所对应的指针未指向文件尾标志时，表示文件中还有数据可读，函数 MORE #n 的值为真，函数 END #n 的值为假；当指针已指向尾标记时，表示文件中已没有数据可读，函数 MORE #n 的值为假，函数 END #n 的值为真。

从数据文件中读取数据时，以上两个函数常用于构成 DO 循环的控制条件，以保证读到文件的末尾。

此外，True BASIC 提供了二个对三类文件都适用的设置文件指针位置的语句。语句格式为：

```
SET #n: pointer begin
```

```
SET #n: pointer end
```

其中 n 为 OPEN 语句中为文件分配的通道号，选用 begin 置指针于文件开头处，选用 end 则置指针于文件的末尾处。

文件刚打开时，指针指在文件开头处，若要为已经存在的数据文件中追加一些数据，必须把指针置于文件末尾处，追加的数据接在原有的数据之后。如果在读文件过程中，需要从头开始重读，应当把指针重新置于文件的开始位置。

置指针于文件开头或文件末尾的语句也可分别写成以下形式：

```
RESET #n: begin
RESET #n: end
```

五、文件删除语句

True BASIC 提供两种删除磁盘文件的语句，即 ERASE 语句和 UNSAVE 语句。

ERASE 语句格式为：

```
ERASE #n
```

语句的作用是删除已打开着的，与指定通道相联系的文件的内容，使之成为一个空文件，并使文件指针指向其开始位置。

一个文件经 ERASE 语句删除全部内容之后，剩下一个文件名，并且还保留原组织形式，仍可对该文件进行打开或关闭操作。需要再把数据写入这个文件时，可按原组织形式相应的写语句写入数据。

UNSAVE 语句格式为：

```
UNSAVE〈文件标识〉
```

语句的作用是删除文件标识所指定的文件。与 ERASE 语句不同，它不仅删除文件的内容，同时也删除文件名。但要注意，被指定删除的文件必须已经存在，且不能处于打开状态，否则，将出错。本语句可作为命令在背景窗使用。

六、ASK（文件查询）语句

文件查询语句有多种格式，利用该语句可以得到与指定通道号 n 相联系的，且已打开的文件的有关信息。表9-2列出了这一语句的多种使用格式及功能。

表9-2

序号	格　　式	功　　能
1	ASK #n: access〈串变量〉	把文件的存取方式（input 或 output 或 outin）赋给串变量

续表9-2

序号	格　　式	功　　能
2	ASK #n: filesize〈数值变量〉	把文件的大小赋给数值变量。对正文文件或字节文件，大小指的是总字节数，即文件长度；对记录文件，大小指的是记录个数
3	ASK #n: margin〈数值变量〉	把正文文件的边界值(行宽)赋给数值变量，若为记录文件或字节文件，其值为零
4	ASK #n: name〈串变量〉	把文件名赋给串变量
5	ASK #n: pointer〈串变量〉	把文件指针位置赋给串变量，位置可以是 begin 或 end
6	ASK #n: organization〈串变量〉	把文件的组织形式(text 或 record 或 byte)赋给串变量
7	ASK #n: record〈数值变量〉	把文件指针的当前位置赋给数值变量。对正文文件，字节文件，其值表示字节数；对记录文件其值表示记录号
8	ASK #n: recsize〈数值变量〉	把记录文件的记录长度赋给数值变量。对正文文件和字节文件，该值为1
9	ASK #n: zonewidth〈数值变量〉	把正文文件数据间的距宽(域宽)赋给数值变量；对记录文件，字节文件，该值为零

第二节　正文文件

一、正文文件的概念与特点

磁盘上正文文件中的数据是由一行行可显示的 ASCII 码字符所组成，每行以回车换行符结束。文件中的各行数据按写入的先后顺序，依次存储在磁盘上。读取正文文件中的数据也只能以行为单位，从文件开头按顺序方式进行。所以，正文文件又称为顺序文件。

由于写在磁盘上正文文件中的数据为 ASCII 码字符，而在内存中，数据的形式为二进制机内代码，因此在读写正文文件时，有一个两种代码的转换过程，然而这个转换过程是由计算机系统自动完成的，用户不必过问。

在打开正文文件的语句中，可选用 access〈存取方式〉，creat〈建立模式〉，organization〈组织形式〉三个选择项。存取方式可选用 output 或 input 或 outin。用 output，只能对文件写，用

input，只能对文件读，用 outin，则对文件既可读，又可写，未选用时，系统默认为 outin。建立模式可选用 new 或 old 或 newold。用 new 表示建立新文件，old 表示使用已存在的文件。newwold 表示文件存在则使用，否则就建立新文件，未选用时，系统默认为 old。组织形式可以缺省，若缺省，建立文件时按第一个对该文件的写语句来确定其组织形式，使用已有的文件时，则按第一个对该文件的读写语句来核对其组织形式，不匹配时则出错。如选用组织形式，则应指定为 text。

需要指出，如以 output 方式输出和 input 方式输入这两种操作方式要进行切换时，则必须先关闭文件，然后再重新打开并选用新的操作方式。

对正文文件写入与读出分别使用 PRINT #和 INPUT #语句及其变形形式。

二、PRINT #（写正文文件）语句

格式1：

```
PRINT #n[,using 〈格式串〉]：表达式1，表达式2，……
```

格式2：

```
MAT PRINT #n[,USING 〈格式串〉]：数组名1，数组名2，……
```

语句的作用是把各表达式的值或各数组中的数据，按指定的格式，写入通道号所对应的正文文件中。

几点说明：

(1) 作为输出项的表达式或数组可以是数值型的，也可以是字符型的。

(2) 如果输出项为数组，其数组元素的排列顺序，与在内存中的存储顺序相同。

(3) 如未选择[，USING 〈格式串〉]项，各输出项之间用逗号或分号分隔，其效果与屏幕显示语句中逗号或分号所起的作用相同。

(4) 如选择了[，USING 〈格式串〉]项，数据将按格式串规定的格式写入正文文件中。

(5) 如果 PRINT #n 之后无冒号和输出项，则向文件写一空行。

〔例9-1〕

```
OPEN #1:name "data-1",create new, org text
READ a,b,c,x$,y$
PRINT #1: a,b,c
PRINT #1: a;b;c
PRINT #1: x$,y$
PRINT #1: x$;y$
PRINT #1: a+b+c,a*b*c
PRINT #1: x$ & y$
DATA 3,5,-2,"True BASIC","PROGRAM"
END

─────────────────────────── ex09-1 ──────────────────

OK. run
```

```
OK
```

运行上述程序，把有关数据写入了正文文件“data-1”中，因为存储的是ASCII码字符，利用OLD命令显示出该文件的内容为：

```
  3                    5                    -2
  3    5    -2
True BASIC          PROGRAM
True BASICPROGRAM
  6                 -30
True BASICPROGRAM
```

由此可见，每执行一次PRINT #语句，就把有关数据写入了相应的正文文件中。写语句内的多项数据，如用逗号分隔，在文件中则以类似屏幕显示中的标准格式排列，如用分号分隔，则以类似屏幕显示中的紧凑格式排列，但数据间的逗号或分号并未写入文件中。从这一点上说，正文文件是屏幕的映象。

三、INPUT #（读正文文件）语句

格式1：

```
INPUT #n：变量名1，变量名2，……
```

格式2：

```
LINE INPUT #n：串变量名1，串变量名2，……
```

格式3：

```
MAT INPUT #n：数组名1，数组名2，……
```

格式4

```
MAT LINE INPUT #n：字符串数组名1，字符串数组名2，……
```

语句的作用是从给定的通道号所对应的正文文件中，以行为单位，一行行地读取数据，提供给有关变量或数组。

其中格式1是从正文文件中依次读取数据提供给各变量，这与键盘输入语句（INPUT语句）为变量提供数据的作用相类似，也需要注意数据和变量的类型一一对应，若使用INPUT #n：串变量，则会自动忽略字符串前后的空格字符；格式2把正文文件中的一整行字符提供给指定的一个串变量，字符串前后的空格不会被忽略；格式3，4是把从正文文件中读取的数据按顺序提供给相应的数组元素。

值得指出的是，如果文件没有打开或打开时选择的存取方式不正确，则不能从该文件中读取数据；如果一行内写入了多项数据，数据项之间没有写入逗号分隔符，多项数据构成了一个ASCII码字符串，回读时，则无法断开成几个数据项，只能使用INPUT #n：串变量或LINE INPUT #n：串变量的读语句，把一整行字符提供给一个串变量。

〔**例9-2**〕 从没有写入逗号分隔符的正文文件“data－1”中读取数据，程序1的读法错误，程序2的读法正确。

程序1

```
OPEN #1: name “data—1”,access input, org text
INPUT #1: a,b,c
PRINT a,b,c
INPUT #1: a1,b1,c1
PRINT a1,b1,c1
INPUT #1: x$,y$
PRINT x$,y$
INPUT #1: x1$,y1$
PRINT x1$,y1$
INPUT #1: m,n
PRINT m,n
INPUT #1: d$
PRINT d$
CLOSE #1
END
```

ex09-2

```
OK. run
String given instead of number.
OK.
```

程序2

```
OPEN #1: name “data—1”,access input, org text
INPUT #1: a$
PRINT a$
INPUT #1: a1$
PRINT a1$
INPUT #1: x$
PRINT x$
INPUT #1: x1$
PRINT x1$
INPUT #1: m$
PRINT m$
INPUT #1: d$
PRINT d$
CLOSE #1
END
```

ex09-02

```
Ok. run
  3                      5                        -2
  3      5      -2
True BASIC           PROGRAM
True BASICPROGRAM
6                        -30
True BASICPROGRAM
Ok.
```

为了回读时，能够把写入一行内的多项数据断开，以便传送给不同的变量，必须在写文件时，在数据项之间写入逗号分隔符。

〔**例9-3**〕 将例9-1程序中有关写文件的语句修改，使逗号分符随数据一起写入文件中，形成名为“data－2”的正文文件。

```
OPEN #1: name "data-2",create newold,org text
ERASE #1
READ a,b,c,x$,y$
PRINT #1: a, ",",b,",",c
PRINT #1: a;",";b;",";c
PRINT #1: x$,",",y$
PRINT #1: x$;",";y$
PRINT #1: a+b+c,",",a*b*c
PRINT #1: x$ & y$
DATA 3,5,-2,"True BASIC","PROGRAM"
END
```

ex09-3

```
OK. run
Ok.
```

用 OLD 命令显示出文件“data－2”的内容，可以看出，逗号分隔符已写入文件中。

```
  3                    ,           5          ,          -2
  3  ,   5   ,-2
True BASIC            ,            PROGRAM
True BASIC,PROGRAM
  6                    ,           -30
True BASICPROGRAM
```

〔**例9-4**〕 从写入了逗号分隔符的正文文件“data－2”中读出数据。

程序为：

```
OPEN #1: name "data-2",access input, org text
INPUT #1: a,b,c
PRINT a,b,c
```

```
INPUT #1: a1,b1,c1
PRINT a1,b1,c1
INUPT #1: x$,y$
PRINT x$,y$
INPUT #1: x1$,y1$
PRINT x1$,y1$
INPUT #1: m,n
PRINT m,n
INPUT #1:d$
PRINT d$
CLOSE #1
END
```

ex09-4

```
Ok. run
 3               5                  -2
 3               5                  -2
True BASIC       PROGRAM
True BASIC       PROGRAM
 6               -30
True BASICPROGRAM
OK.
```

通过上述例子看出，如果写正文文件时，一行内的多项数据间均写入了逗号分隔符，回读时使用的 INPUT #语句中可以有多个变量，其个数和类型与一行内写入的数据的个数和类型要一一对应，则能把多项数据断开，提供给不同的变量。

因此，在写正文文件的时候，就要考虑该文件如何回读，即要考虑 PRINT #语句与 INPUT #语句的配合使用问题。如果对文件中的数据不需要分项读出处理，则数据项间不必写入逗号分隔符，回读时可使用 INPUT #n:串变量或 LINE INPUT #n: 串变量形式的读语句；如果对文件中的数据需要分项读出进行具体运算或操作，则数据项之间应写入逗号分隔符，回读时使用带有多个相应类型变量的 INPUT #语句。

四、设置行宽，域宽语句

正文文件以行为单位读写，为了控制文件写入的格式，True BASIC 提供了设置行宽和域宽的语句。

语句格式：

SET #n: margin (数值表达式)

SET #n: zonewidth (数值表达式)

语句的作用是分别设置正文文件行的宽度和数据项的域宽。

其中 n 为 OPEN 语句中分配的通道号，margin 为行宽说明，其后的数值表达式的值为80～32767之间的整数，zonewidth 为数据项域宽说明，其后数值表达式的值为1～60之间的整数。

写正文文件时，若未使用设置行宽或域宽的语句，系统则默认行宽为80位，域宽为16位。

五、向打印机输出数据

True BASIC 把打印机作为一个特殊的正文文件对待，文件名为 PRINTER，用打印机输出数据，即相当于向 PRINTER 文件写入数据。在打印前，先用以下语句打开打印机：

```
OPEN #n: PRINTER
```

在此语句中，不要使用 name〈文件标识〉，access〈访问方式〉，create〈建立模式〉，organization〈组织形式〉等选择项。

在执行了打开打印机语句之后，就可以使用写正文文件语句，将数据从打印机上输出。

对打印机不允许使用设置文件指针语句。

〔**例9-5**〕 将若干个学生的学号，姓名，性别（设 m 为男性，w 为女性），年龄送打印机打印。

程序为：

```
! print number name sex age of student
OPEN #1: printer
LET x$="%%%%%%  <##########    #      ##"
PRINT #1: "number    name    sex    age"
PRINT #1: "----------------------"
DO while more data
   READ num,name$,sex$,age
   PRINT #1,using x$:num,name$,sex$,age
LOOP
CLOSE #1
DATA 526106,Li ming,m,19
DATA 526112,Wang dung,m,20
DATA 526120,Chen hong,w,21
DATA 526133,Zheng san,m,18
END
```

运行上述程序，在打印机上输出的结果为：

```
number     name      sex    age
----------------------
526106    Li ming     m      19
526112    Wang dung   m      20
526120    Chen hong   w      21
526133    Zheng San   m      18
```

7

六、正文文件应用举例

建立和使用一个正文文件的一般步骤为：

(1) 使用 OPEN 语句打开文件，在打开语句中为指定的文件分配一个通道号，并选用 access，create，organizaion 项说明文件访问方式，建立模式及组织形式。

(2) 对文件进行写入或读出操作，均以行为单位，顺序进行。写入数据只能从文件开头（建立文件时）或文件末尾（追加数据时）处进行，可设置行宽和域宽来控制写入的格式，读出数据必须从开头处进行，一次读出一行，而不能将一行数据部分读出。

(3) 文件操作结束后，及时关闭文件。

〔**例9-6**〕 将一个学生的学号，姓名及三门课成绩写入一个正文文件。

程序为：

```
! create score fire
OPEN #1: name "score",org text, create new
PRINT #1: 781205
PRINT #1: "Wang lin"
PRINT #1: 92
PRINT #1: 87
PRINT #1: 90
CLOSE #1
END
```

———————————————— ex09-6 ————————————————

```
OK. run
OK.
```

〔**例9-7**〕 将上例建立的正文文件“score”中的内容读出并显示出来。

程序为：

```
! read score fire
OPEN #1: name "score"
INPUT #1: n
INPUT #1: name $
INPUT #1: s1
INPUT #1: s2
INPUT #1: s3
PRINT n; "     ";name $ ;"     ";s1;"     ";s1;"     ";s3
CLOSE #1
END
```

———————————————— ex09-7 ————————————————

```
OK. run
```

```
781205          Wang lin          92          87          90
OK.
```

写“score”文件时，每个数据单独写了一行，所以从此文件中可以将数据一个一个地读出，写入的数值型数据读出到一个数值变量中；写入的字符串读到串变量中。这里由系统自动实现了从 ASCII 码字符到数值的变换。由于一个人的数据作为几行写入文件中，如果文件中有多人的数据，就必须严格按写入时的顺序及类型对应读出，否则，容易产生错误与混乱。

〔**例9-8**〕 将某商店销售的若干种商品的代号，名称，单价，销售数量及金额写入一个正文文件中。其程序中：

```
! create sale fire
OPEN #2: name "sale",create new, org text
DO until end data
   READ number, name$ ,price, quantity, amount
   PRINT #2: number,name$ ,price,quantity,amount
LOOP
CLOSE #2
DATA 101,YELLOW RIVER COLOR TV,2800,12,33600
DATA 102,GREAT WALL COLOR TV, 2500,10,25000
DATA 201,PANDA RADIO,48.60,18,874.80
DATA 202,REDLAMP RADIO,27.76,20,555.20
DATA 301,ORIOLE RECORDER,135.00,15,2025
END
```

————————————————————— ex09-8 —————————————

```
OK. run
OK
```

〔**例9-9**〕 将上述已建立的正文文件“sale”的内容读出并显示出来。程序如下：

```
! print sale file
OPEN #1: name "sale",access input, org text
DO while more #1
   INPUT #1: x$
   PRINT    x$
LOOP
CLOSE #1
END
```

————————————————————— ex09-9 —————————————

```
OK. run
101               YELLOW RIVER COLOR TV              2800           12
33600
```

```
102                 GREAT WALL COLOR TV                2500            10
25000
201                 PANDA RADIO      48.6              18              874.8
202                 REDLAMP RADIO   27.76              20              555.2
301                 ORIOLE RECORDER                    135             15
2025
Ok.
```

这里从"sale"文件中一次读出一行字符传送给内存中的串变量 x$，并显示之。执行程序后发现，显示在屏幕上的数据"YELLOW RIVER COLOR TV"和"GREAT WALL COLOR TV"因超过默认的域宽16位而占据了两个标准位置，造成该商品的五项数据不能显示在一行，整个输出不清晰。究其原因是，在建立该文件时，以默认的行宽80位，域宽16位写入，相应商品的五项数据不是写在一行内，而是写了两行。为了使一种商品的数据显示在一行上，可用 SET #语句调整行宽和域宽。相应的程序为：

〔**例9-10**〕

```
! create sale fire
OPEN #2: name "sale",create newold, org text
ERASE #2
SET #2: margin 120
SET #2: zonewidth 8
DO until end data
    READ number, name $ ,price, quantity, amount
    PRINT #2: number,name $ ,price, quantity,amount
LOOP
DATA 101, YELLOW RIVER COLOR TV,2800,12,33600
DATA 102,GREAT WALL COLOR TV,2500,10,25000
DATA 201,PANDA RADIO,48.60,18,874.80
DATA 202,REDLAMP RADIO,27.76,20,555.20
DATA 301,ORIOLE RECORDER, 135.00,15,2025
SET #2: pointer begin
DO while more #2
    INPUT #2: x$
    PRINT     x$
LOOP
CLOSE #2
END
```

———————————————— ex09-10 ————————————————

```
Ok. run
101    YELLOW RIVER COLOR TV       2800      12         33600
```

```
102    GREAT WALL COLOR TV          2500      10        25000
201    PANDA RADIO          48.6      18     874.8
202    REDLAMP RADIO        27.76     20     555.2
301    ORIOLE RECORDER      135       15     2025
OK.
```

〔**例9-11**〕 建立一个职工工资文件，每个职工有编号，姓名，年龄，月基本工资四项数据，将每人的数据写入文件中的一行，数据由 DATA 语句提供，其程序为：

```
! create wage file
OPEN #1: name "wage",create newold, org text
LET x$ ="#### ,<############### ,## ,####.##"
DO while more data
   READ number,name$ ,age,pay
   PRINT #1,using x$ : number,name$ ,age,pay
LOOP
CLORE #1
DATA 301,LI MING, 50, 160.00
DATA 302,WANG XIAO DUNG,32,105.40
DATA 303,ZHANG HONG,25,89.50
DATA 304,FAN YUAN,22,82.00
DATA 305,YI LI HUA,38,113.00
END
```

———————————————————— ex09-11 ————————————————————

```
OK. run
OK
```

上述例子，通过在格式字符串中把逗号放在非特殊符号（此处为空格）中间的方法，执行带 USING 短语的 PRINT #语句，实现在数据项之间写入逗号分隔符。

〔**例9-12**〕 从上例建立的正文文件"wage"中检索出月基本工资在100元以下的职工，并把这些职工的各项数据存入一个低工资文件"low-wage"中。其程序为：

```
! print pay<100
OPEN #1:name"wage",access input, org text
OPEN #2:name"low-wage",access output,create newold, org text
ERASE #2
LET x$ ="#### ,<############### ,## ,####.##"
PRINT "No. ","name","age","pay"
PRINT repeat$ ("-",55)
DO while more #1
   INPUT #1: number, name$ ,age, pay
   IF pay < 100 then
```

```
        PRINT number, name $ ,age, pay
        PRINT #2, using x $ : number, name $ ,age,pay
    END IF
LOOP
CLOSE #1
CLOSE #2
END

------------------------------------------------------------ ex09-12 ------------------------------
Ok. run
No.            name            age            pay
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
   303        ZHANG HONG      25             89.5
   304        FAN YUAN        22             82
OK.
```

七、显示格式文件

在 True BASIC 中，用 SAVE 命令存入磁盘的程序和数据，由于存储的是 ASCII 码字符，且每行以回车换行符结束，所以它们也是正文文件。这种特殊的正文文件称为显示格式文件，它既能用于存储程序，又能用于存储数据。文件内容通过键盘输入，并能进行屏幕编辑，然后用 SAVE 命令，以指定文件名写入磁盘。如果显示格式文件用于存储数据，与一般正文文件不同，数据不是通过 PRINT #（写正文文件）语句记入磁盘，而是通过键盘提供，用 SAVE 命令写入。数据因从键盘输入，只能为数值或字符串，而不能是变量或表达式。需要对显示格式文件中的数据进行处理时，可在打开文件后，采用与读一般正文文件一样的读语句，从显示格式文件中读取数据。

〔**例9-13**〕 将10名女生的号码，姓名和100米短跑成绩（秒），存入名为“mark”的显示格式文件中。

```
107,Chen hong,14.2
124,Zhou ting,14.0
231,Ding xiao hua, 13.4
240,Li ling ling, 13.7
205,Wang li li, 13.9
301,Yang ming,13.5
312,Ma xiao ping, 13.6
408,Fan yuan yúan, 13.2
517,Zhzn dan dan,13.3
626,Liu jian,13.8

------------------------------------------------------------ Untitled ------------------------------
```

```
Ok. save mark
  10 lines saved.
OK.
```

这里通过键盘在编辑窗输入全部数据，然后在背景窗输入 save mark 并按回车键，在当前盘上即生成了一个名为“mark”的显示格式文件，键入的全部数据及数据间的逗号分隔符都被写入了该文件。

欲把“mark”文件中全部数据列在屏幕上，可在背景窗打入命令 OLD mark 实现，也可以运行程序来实现。

〔例9-14〕

```
OPEN #1: name "mark"
DO while more #1
   LINE INPUT #1: x$
   PRINT x$
LOOP
END
```

———————————————————— ex09-14 ————————————————————

```
OK. run
107,Chen hong, 14.2
124,Zhou ting,14.0
231,Ding xiao hua,13.4
240,Li ling ling,13.7
205,Wang li li,13.5
301,Yang ming,13.5
312,Ma xiao ping,13.6
408,Fan yuan yuan,13.2
517 Zhan dan dan,13.3
626,Liu jian,13.8
OK.
```

〔例9-15〕 从“mark”文件中检索出运动成绩不超过13.5秒的学生，输出她们的号码，姓名及运动成绩。其程序为：

```
OPEN #1: name "mark"
PRINT "NO              NAME              MARK"
PRINT "********************************"
DO while more #1
   INPUT #1: n,name$,m
   IF m<=13.5 then
      PRINT n, name$,m
   END IF
```

```
LOOP
CLOSE #1
END
```

ex09-15

```
Ok. run
NO              NAME              MARK
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
  231         Ding xiao hua          13.4
  301         Yang ming              13.5
  408         Fan yuan yuan          13.2
  517         Zhan dan dan           13.3
OK.
```

〔**例9-16**〕 有四个百货商店，它们当天售出商品 A，B，C 的数量和单价如表9-3和表9-4所示。

表9-3　　　　**商品销售日报表**

商店名 售出数量 商品名	A	B	C
第一百货商店	56	64	88
第二百货商店	126	42	64
第三百货商店	82	193	151
第四百货商店	205	126	174

表9-4

商品名	单价
A	1286
B	745
C	521.6

试编程序，分别求出各百货商店当天的总销售额。

设商品出售数量，商品单价，各商店当天总销售额分别存于数组 sale(4，3)，price(3)，amount(4)中。

如果把上述两个表格中的数据已依次写入名为“store”的显示格式文件里，即：

```
56,64,88,126,42,64,
82,193,151,205,126,174
1286,745,521.6
```

```
——————————————————————— Untitled ———————————————
OK. save store
  3 lines saved.
Ok.
```

数组 sale，price 的各元素的值从显示格式文件“store”中读取，然后计算出数组 amount 各元素的值并输出。

需要注意的是：使用 MAT INPUT ＃语句读取数据时，提供给一个数组的一批数据在文件中既可以放在一行内，也可以放在几行内，数据项之间应有逗号分隔，如果放在几行内，在每一行（其最后一行除外）的结尾处要有一个逗号。提供给不同数组的数据则不能放在一行内。

程序为：

```
! calculate sales for several stores
DIM sale(4,3),price(3),amount(4)
OPEN #1: name "store"
MAT INPUT #1: sale, price
PRINT "     mat sale"
MAT PRINT sale
PRINT "     mat price"
MAT PRINT price
PRINT "     mat amount"
MAT amount =sale * price
FOR i=1 to 4
    PRINT "amount(";i;")=";
    PRINT USING "# # # # # # # #. # #": amount(i)
NEXT i
CLOSE #1
END
```

```
——————————————————————— ex09-16 ———————————————
OK. run
     mat sale
56              64             88
126             42             64
82              193            151
205             126            174
     mat price
1286            745            521.6
     mat amount
amount(1)= 165596.80
amount(2)= 226708.40
amount(1)= 327998.60
amount(1)= 448258.40
OK.
```

从上面几个例子看出，显示格式文件为用户提供了一种更为简便的保存数据的手段。利用这种手段，还可以把大批的报表数据存入磁盘，生成正文文件。这些数据不仅 True BASIC 能处理，而且其他语言环境（如 DBASE）也能处理。这种文件可成为 True BASIC 与其他软件环境传递数据的接口文件。

第三节　记录文件

一、记录文件的概念与特点

记录文件是记录的有序集合，是 True BASIC 提供的另一种组织形式的文件。记录文件明显具有自己的特点及处理方式。

(1) 记录文件的每个记录有唯一的数据项，并且占有相同大小的存储空间（记录长度）。若要把相关的多个数据写入一个记录，可将其中的数值型数据利用特定的字符串函数转换为串，然后将这多个数据连接成为一个新的字符数据项。在一个记录文件中，各记录可以存放数值型数据，也可以存放字符型数据，还可以是两种类型的混合存放。

(2) 记录从1开始，按自然数的递增顺序编号，对记录文件的读写操作及管理都是以记录为单位进行的。可以按指定的记录号随机地读写文件中的某一个记录，所以记录文件也称为随机文件。

(3) 记录文件中的数据都是以机内二进制代码形式存储在磁盘上的，因此在对记录文件读写时，不存在两种代码的转换。

(4) 打开记录文件的语句中，不用指定访问方式，因为记录文件打开后，既可以向文件写入，也可由文件读出。建立模式的选择与正文文件的相同。组织形式项可缺省，若缺省，新建文件时，由对该文件的第一个写语句来确定文件的组织形式，读写已存在的文件时，由对该文件的第一个读写语句来核对其组织形式，不匹配时，则出错；若不缺省，则应指定为 record。

(5) 记录文件的记录长度在建立文件时确定，不指定时，系统默认其为80个字节，不为80个字节时，必须指定具体值。指定记录长度的方法有两种，一种是在 OPEN 语句中使用 recsize 选择项，如：

```
OPEN #1: name "file",create new, org record, recsize 32
```

则指明文件 file 的记录长度为32字节。

另一种方法是在打开语句之后，第一个写语句之前，使用设置语句来设置记录长度。这种设置语句的格式为：

```
SET #n: recsize〈数值表达式〉
```

如：SET #1: recsize 32

也表明相应文件的记录长度为32字节。这两种方法，可任选其一。

记录长度是按字节来计数的，一个数值型数据占8个字节，字符型数据中的每个字符占1个

字节。由于记录按定长的字节来存储，所以记录中数据项的实际长度小于规定的记录长度时，系统将自动用空格填充剩余位，以达到规定的长度。

记录文件的记录长度最长不得超过2^{24}（大约16兆）字节。

需要注意的是，只能对新建立的记录文件设置记录长度，而对磁盘上已经存在的记录文件（空文件除外），不允许重新设置记录长度。

（6）记录文件的写入用 WRITE #语句，读出用 READ #语句。

二、WRITE #（写记录文件）语句

向记录文件写入数据，应使用 WRITE #语句，其格式为：

格式1

```
    WRITE #n：表达式1，表达式2，……
```

格式2

```
    MAT WRITE #n：数组名1，数组名2，……
```

其中 n 为打开语句中给文件分配的通道号，各表达式或各数组既可以是数值型的，也可以是字符型的，彼此间用逗号隔开。

执行 WRITE #语句时，系统依照当前指针的位置，把一个数据项写入一个相应的记录中，每写完一个记录，指针向下移动一次。格式1是把各表达式的值依次写入各记录中；格式2是把数组元素的值按其存储顺序依次写入到相应的各记录中。

通过移动指针，可对记录文件进行覆盖重写，或者在文件末尾写入，以追加新的记录。如果文件没有打开，或打开的是一个正文文件时，用 WRITE #语句写入，将会出错。

〔**例9-17**〕 建立一个记录文件，把若干种杂志的代号，名称，定价写入文件中。

程序中：

```
! create magazine file
OPEN #1: name "magazine",create new, org record
SET #1: recsize 15
DO while more data
   READ n$,name$,price
   WRITE #1: n$,name$,price
LOOP
CLOSE #1
DATA 1-25,computer world,1.52
DATA 1-38,dbase,3.50
DATA 1-56,basic program, 2.86
DATA 1-132,China computer, 1.20
DATA 1-48,software, 3.90
END
```

———————————————— ex09-17 ————————————————

```
OK. run
OK.
```

执行上述程序，每次由 READ 语句从数据区读三项数据，然后由 WRITE # 语句将这三项数据写入文件，形成三个记录。结果在当前盘上建立了一个名为 magazine 的记录文件，文件中共有15个记录。

记录文件中的数据由于是以机内代码形式存储，因此不能以易读的格式在屏幕上直接显示。

三、READ #（读记录文件）语句

已建立的记录文件中的数据只有用读语句读出后，才能为用户所利用，读语句的格式为：

格式1

```
READ #n：变量名1，变量名2，……
```

格式2

```
MAT READ #n：数组名1，数组名2，……
```

语句的作用是从记录文件中读取记录内容，提供给相应的变量或数组。

这里 n 为打开文件语句中指定的通道号，多个变量名或数组名彼此间用逗号分隔，变量或数组的类型要与读出的数据的类型相一致。

执行 READ # 语句时，将从指针当前位置开始读取，每读一个记录后，指针自动移到下一个记录，如果一个 READ # 语句里有多个变量名，则从文件中读取相应个数的记录。如果相应的文件没有打开或打开的是一个正文文件，使用 READ # 将会出错。

〔**例9-18**〕 从上述建立的“magazine”记录文件中读出全部记录，并在屏幕上显示出来。

程序为：

```
! read "magazine" file
OPEN #2: name "magazine"
DO while more #2
    READ #2: k$,x$,p
    PRINT k$,x$,p
LOOP
CLOSE #2
END
```

```
——————————————————————ex09-18——————————————————————
OK. run
1-25        computer world        1.52
1-38        dbase                 3.5
1-56        basic program         2.86
1-132       China computer        1.2
1-48        software              3.9
Ok.
```

这里利用 READ #语句一次从文件中读出三个记录，再通过屏幕显示语句，把这三个记录显示在一行内。

四、设置记录文件指针语句

如前所述，对记录文件的读写是按指定的记录进行的。每个打开的文件隐含一个指针，指针指向的记录称为当前记录，任何时候只能对当前记录进行读写操作。可以使用前面所述的对三类文件都适用的设置文件指针语句，将指针指向文件开头(即第1个记录)或结尾(即最后一个记录之后)，除此之外，为实现随机读写，True BASIC 还提供了三个适用于记录文件和字节文件的设置指针位置语句。使用时可将文件指针指向所需要读写的位置。

这三个语句的格式为：

```
SET #n: pointer same
```

```
SET #n: pointer next
```

```
SET #n: record rn
```

它们可以分别写为：

```
RESET #n: same
RESET #n: next
RESET #n: record rn
```

其中 n 为打开文件语句中指定的通道号，选择 same 表示指向当前记录的上一个记录；选择 next 表示指向当前记录的下一个记录。若选择 record rn，表示将指针指向第 rn 个记录。

需要注意的是：

(1) 设置指针位置之前，相应的数据文件必须是打开的。

(2) 每读写完一个记录之后，指针自动移动，指向下一下记录。

(3) 记录文件刚打开时，指针自动指向第一个记录，若对最后一个记录读写完，指针则指向文件尾标志。

例9-18中未采用指针设置语句，则从记录文件“magazine”打开时的默认位置——第1个记录开始，依次读取全部记录中的数据。

下面的程序是利用指针设置语句，读取该文件中的部分记录。

〔例9-19〕

```
OPEN #2: name "magazine"
SET #2: record 4
READ #2: k$,x$,p
PRINT k$,x$,p
RESET #2: NEXT
RESET #2: NEXT
RESET #2: NEXT
READ #2: k$,x$,p
PRINT k$,x$,p
```

```
RESET #2: SAME
RESET #2: SAME
RESET #2: SAME
READ #2: k$,x$,p
PRINT k$,x$,p
RESET #2: BEGIN
READ #2: K$,x$,p
PRINT K$,x$,p
CLOSE #2
END
```

——————————————————————— ex09-19 ———————————————

```
OK. run
1-38             dbase            3.5
1-132            China computer   1.2
1-132            China computer   1.2
1-25             computer world   1.52
OK.
```

五、具有多数据项的记录文件

在上节建立的记录文件“magazine”中，每个记录只存放一个原始数据，不管实际数据的长短，皆占用规定的记录长度。而实际数据因类型和含义不同，长度差别可能是很大的，比如杂志名最长需占15个字节，而杂志代号不超过6个字节，定价占8个字节，但记录长度只能按需要的最长字节数来设置，即15字节，这样在存储杂志代号和定价时，就浪费很多的存储空间。另外，一种杂志的三项数据须放在三个记录中。比如第一种杂志的代号，名称，定价为第1,2,3号记录，第二种杂志的代号，名称，定价为第4,5,6号记录等等，记录号与记录内容的对应关系不够清晰，给查询和处理某一项数据带来不便。

解决的办法是将含义相关的多项原始数据合并，装配成一个新的字符串，每项原始数据在这个字符串中都有固定的位置，把该字符串写入一个记录中，字符串的长度即为记录长度。用这种方式建立的记录文件称为多数据项记录文件。

〔**例9-20**〕 将杂志代号，名称，定价等数据装配成一个字符串，以建立多数据项记录文件。设代号占6个字节，名称占15个字节，定价占8个字节，文件名为“magazi”，相应程序为：

```
! create record fire
OPEN #1: name "magazi", org record, create newold
SET #1: recsize 29
ERASE #1
DO while more data
   READ n$,name$,price
   LET n$=n$ & repeat$("  ",6-len(n$))
```

```
    LET name $ =name $  & repeat $ ("    ",15-len(name $ ))
    LET p $ =num $ (price)
    LET s $ =n $  & name $  & p $
    WRITE #1: s $
LOOP
CLOSE #1
DATA 1—25, computer world, 1..52
DATA 1—38,dbase, 3. 50
DATA 1—56,basic program,2. 86
DATA 1—132,China computer,1. 20
DATA 1—48,software,3. 90
END
```

———————————————————— ex09-20 ————————————

```
OK. run
OK.
```

程序中在每提供一种杂志的代号 n $，名称 name $，单价 price 的值之后，接着执行以下语句：

```
LET n $ =n $  & repeat $ ("    ",6—len(n $ ))
LET name $ =name $  & repeat $ ("    ",15—len(name $ ))
LET p $ =num $ (price)
```

其目的是使杂志代号不足6位时，右端自动加空格补足6位，杂志名不足15位时，右端自动加空格补足15位，将定价 price 转换成一个8字节的字符串，然后将这三个字符串数据项连接成一个新的字符串项 s $，再把 s $ 作为一个记录写入文件中。

从多数据项记录文件中读取数据，仍以记录为单位进行。将一个记录的内容读给一个字符串项后，必须把这个字符串项再分解还原成多个数据项，从而便于实现对数据项的查找和处理。

〔**例9-21**〕 从上面已建立的数据记录文件“magazi”中，查询名为“dbase”的杂志，输出其有关数据。其程序为：

```
! input record file
OPEN #1: name "magazi"
DO while more #1
    READ #1: s $
    LET n $ =s $ [1:6]
    LET name $ =s $ [7:21]
    LET price=num(s $ [22:29])
    LET name $ =rtrim $ (name $ )
    IF name $ ="dbase" or name $ ="DBASE" then
       PRINT n $ ,name $ ,price
    END IF
LOOP
CLOSE #1
END
```

ex09-21

```
Ok. run
1－38            dbase            3.5
Ok.
```

程序中使用了以下语句：

```
LET n$ =s$[1:6]
LET name$ =s$[7:21]
LET price=num(s$[22:29])
```

即实现字符串的分解，并对数值型数据项进行还原。

六、记录文件应用举例

建立和使用一个记录文件的一般步骤是：

（1）用 OPEN 语句打开文件。在打开语句中为文件分配一个通道号，不用指定访问方式，组织形式不缺省时应指定为 record，建立模式的选择同正文文件，对新建立的记录文件，若记录长度不为80字节时，应在打开语句或 SET #语句中指定记录长度。

（2）按文件指针的当前位置，以记录为单位，对文件进行读写及其他操作，写入记录用 WRITE #语句，读出记录用 READ #语句。

（3）对文件操作结束后，及时关闭文件。

〔**例9-22**〕 建立一个商品库存管理系统，可实现如下功能：

建立商品库存文件

查询文件

库存商品数据的更新（即追加，修改，删除记录）

显示库存商品清单

按照结构化程序设计思想，整个系统由一个主控模块和多个功能模块组成。功能模块分别设计成外部子程序，并把这些子程序存于名为“TAMS”（即 Trade Articles Management System）的库文件中。在主控模块程序中，提供功能选择菜单，用户在菜单提示下，根据需要输入相应的选择，去调用库文件中不同的子程序，以实现对记录文件的不同操作。

主控模块

在程序中提供以下功能选择菜单：

（1）建立一个商品文件
（2）查询商品文件
（3）追加新记录
（4）修改记录
（5）删除记录

(6)显示全部记录
(7)退出

主程序为：

```
! main pragram
LIBRARY “TAMS”
DO
      CLEAR
      PRINT tab(10);“1”;tab(15);“create a aticles file”
      PRINT
      PRINT tab(10);“2”;tab(15);“search articles file”
      PRINT
      PRINT tab(10);“3”;tab(15);“expand new record”
      PRINT
      PRINT tab(10);“4”;tab(15);“modify record”
      PRINT
      PRINT tab(10);“5”;tab(15);“delete record”
      PRINT
      PRINT tab(10);“6”;tab(15);“display all records”
      PRINT
      PRINT tab(10);“7”;tab(15);“end”
      INPUT prompt “function choice:”:x
      SELECT CASE x
      CASE 1
            CALL createfile
      CASE 2
            CALL search
      CASE 3
            CALL expand
      CASE 4
            CALL modify
      CASE 5
            CALL delete
      CASE 6
            CALL display
      CASE 7
            PRINT “--------------------------------”
            PRINT “                  bye-bye!               ”
            PRINT “--------------------------------”
            EXIT DO
```

```
        CASE else
             PRINT "wrong ! again input a choice"
             PAUSE 10
        END SELECT
    LOOP
    END
```

主程序中，必须在调用各子模块的语句前，设置 LIBRARY “TAMS”语句。运行该主程序时，首先在屏幕上列出功能菜单，用户键入1～6之间选择码，则转去调用相应的子程序，然后返回，重新显示菜单，如选择码为7，则退出文件操作，返回“OK”状态。

建立商品库存文件模块

建立一个商品库存记录文件，其记录由商品的编号，名称，单价，数量，存放地点五项数据组成。设编号 no$ 占5字节，名称 name$ 占15字节，单价 price 占8字节，数量 quantity 占8个字节，存放地点 place$ 占6字节，则记录长度为42字节。

考虑到建立文件和向文件追加记录时都需要多次写一个记录，可把完成写一个记录的程序段设计成一个子程序 writerec，其中使用通道号作为形参。在建立文件和追加记录模块的程序中调用该子程序，调用时，形参通道号将被实参通道号所替换。

下面分别是建立文件子程序和写一个记录子程序：

```
    ! create stock articles of trade
    SUB createfile
        OPEN #1: name"stock",create new, org record
        SET #1: recsize 42
        INPUT prompt "total of record:":n
        FOR i=1 to n
            CALL writerec(#1)
        NEXT i
        CLOSE #1
    END SUB

    ! write a record
    SUB writerec(#999)
        INPUT prompt "No. of article :": no$
        INPUT prompt "name of article:": name$
        INPUT prompt "price of article:": price
        INPUT prompt "quantity of article :": quantity
        INPUT prompt "place of article :": place$
        LET no$ =no$ & repeat$("  ",5-len(no$))
        LET name$ =name$ & repeat$("  ",15-len(name$))
        LET place$ =place$ & repeat$("  ",6-len(place$))
        LET p$ =num$(price)
        LET q$ =num$(quantity)
        LET s$ =no$ & name$ & p$ & q$ & place$
        WRITE #999:s$
    END SUB
```

子程序 writerec 中，通过一组 INPUT 语句，从键盘输入一种商品的五项数据，然后按固定格式把这五项数据装配成一个新的字符串 s$ 再执行 WRITE # 语句，即生成一个记录。

在建立文件子程序 createfile 中，先从键盘输入所建文件记录总个数(即商品种数)，然后借助 FOR 循环实现调用子程序 writerec 相应的次数，将在磁盘上建立一个名为“stock”的记录文件，有多少种商品，则有多少个记录。

查询文件模块

第六章已经讨论了各种查询算法的设计，下面给出以商品名为待查项，按顺序查询算法查询某种商品的子程序：

```
    ! search articles of trade
    SUB search
        OPEN #1: name“stock”
        INPUT prompt “name of article:”;m$
        LET m$ =m$ & repeat$(“  ”,15-len(m$))
        LET flag=0
        DO while more #1
           READ #1: s$
           IF m$ =s$[6:20] then
              LET flag=1
              LET no$ =s$[1:5]
              LET name$ =s$[6:20]
              LET price=num(s$[21:28])
              LET quantity=num(s$[29:36])
              LET place$ =s$[37:42]
              PRINT no$ ;“  ”;name$ ;“  ”;
              PRINT price;“  ”;quantity;“  ”;place$
              EXIT DO
           END IF
        LOOP
        IF flag=0 then PRINT “there is not such article.”
        DO
        LOOP until key input
        GET KEY x
        CLOSE #1
    END SUB
```

这里假设一种商品名唯一地对应一个记录，所以一旦找到，在输出该商品的各项数据之

后，即可跳出循环，结束查找，如果遍历所有记录仍未找到(flag＝0时)，则输出无此商品的信息。

显然，如果待查项与记录没有唯一对应的关系，则必须遍历文件的所有记录，不能因检索成功而中途跳出循环。

文件更新模块

为了使文件中的数据反映客观情况的变化，需要对数据文件的内容进行更新，记录文件的更新操作包括：

追加新的记录；

修改文件中某些记录；

删除文件中已无保留必要的记录。

与正文文件相比，记录文件更便于查询及各种更新操作。

·追加新记录子模块

往已经存在的记录文件中追加若干新记录，其算法为：首先打开文件，并将文件指针置于文件的末尾，然后利用循环结构，从键盘输入数据并经 WRITE #语句写入文件，结束追加，关闭文件。

实现此算法的子程序为：

```
! expand new records
SUB expand
    OPEN #1: name "stock"
    SET #1: pointer END
    DO
        CALL writerec(#1)
        PRINT "* * * * * expanded a record * * * * *"
        INPUT prompt "continue (y/n)": x $
        IF x $ ="N" OR x $ ="n" then EXIT DO
    LOOP
    CLOSE #1
END SUB
```

在执行上述子程序时，又调用了实现写一个记录的子程序 writerec，每调用一次，追加一个记录。

·修改记录子模块

修改记录算法：文件打开之后，根据待查项的值先查找欲修改的记录，查找成功，则在内存中对该记录的某些数据项进行修改，然后使指针重新指向该记录，并用 WRITE #语句将修改后的记录值写入原记录，实现记录新值覆盖旧值。若未找到要修改的记录，则给出“未找到”的信息。若要修改多个记录，则应重复上述过程。但要注意，每次重复均应重新设置指针位置为 BEGIN。结束修改，关闭文件。

其子程序为：

```
! modify record of file
SUB modfy
```

```
OPEN #1: name “stock”
DO
    SET #1: pointer BEGIN
    INPUT prompt “name of article to modify :”: m$
    LET flag=0
    DO while more #1
        READ #1: s$
        IF rtrim$(s$[6:20])=m$ then
            LET flag=1
            LET no$=s$〔1:5〕
            LET name$=s$[6:20]
            LET price=num(s$[21:28])
            LET quantity=num(s$[29:36])
            LET place$=s$[37:42]
            PRINT no$;“   ”;name$;“   ”;
            PRINT price;“   ”;quantity;“   ”;place$
            LINE INPUT prompt “change no:”: no1$
            IF no1$=“” then LET no1$=no$
            LINE INPUT prompt “change name:”: name1$
            IF name1$=“” then LET name1$=name$
            LINE INPUT prompt “change price:”: p1$
            IF p1$=“” then
                LET price1=price
            ELSE
                LET price1=val(p1$)
            END IF
            LINE INPUT prompt “change quantity:”:q1$
            IF q1$=“” then
                LET quantity1=quantity
            ELSE
                LET quantity1=val(q1$)
            END IF
            LINE INPUT prompt “change place:”: place1$
            IF place1$=“” then LET place1$=place$
            PRINT no1$;“   ”;name1$;“   ”;
            PRINT price1;“   ”;quantity1;“   ”;place1$
            LET no1$=no1$ & repeat$(“   ”,5-len(no1$))
            LET name1$=name1$ & repeat$(“   ”,15-len(name1$))
            LET p1$=num$(price1)
            LET q1$ =sum$(quantity1)
            LET place1$ =place1$ & repeat$(“   ”,6－len(place1$))
            LET s$ =no1$ & name1$ & p1$ & q1$ & place1$
            RESET #1: same
            ASK #1 : record n
            PRINT “record:”;n
            WRITE #1: s$
            EXIT DO
          END IF
      LOOP
      IF flag=0 then PRINT “haven't found!”
      INPUT prompt “modify another record ?(y/n)”:x$
      IF x$ =“N” or x$ =“n” then EXIT DO
  LOOP
  CLOSE #1
END SUB
```

调用此子程序时，根据键入的商品名，查找相应的记录，若找到，将该记录的内容显示在屏幕上，核对后，可修改记录中的部分或全部数据项，其中不必修改的项，在相应的提示信息之后只要按回车键(意味着输入一个空串)即可。若未找到，则显示出“找不到”的信息。

·删除记录子模块

从文件中删除记录的实现方法有两种，即物理删除和逻辑删除。逻辑删除是指被删除记录仍占有原存储位置，只是在记录中打上特定的标志，如：将商品号改为“－1”，商品名改为“END”等。以后按相应待查项查询此文件时，找不到该记录，这实际上就起到删除指定记录的作用。物理删除是指将记录所占据的存储区域收回，或者说把它从文件中取消。下面介绍记录文件物理删除记录的算法。

首先打开两个文件，一个是存储有被删除记录的原文件，另一个则是用来实现“誊写”操作的新文件。新文件应按 create new 方式打开，其记录长度，组织形式与原文件相同。接着按顺序算法对原文件的各个记录进行测试，若是欲删除的记录，则不往新文件中“誊写”，否则，原封不动地“抄”写到新文件中去，这一过程直到原文件无记录为止。然后用 EARSE # 语句删除原文件中的全部记录，再把新文件中的记录“抄”回到原文件。以上操作结束之后，关闭这两个文件，并利用 UNSAVE 语句删除新文件。

实现以上算法的子程序为：

```
! delete a record of file
SUB delete
    OPEN #1: name “stock”
    OPEN #2: name “stock-d”,org record, create new
    SET #2: recsize 42
    INPUT prompt “name of article to delete :”: m$
    LET flag=0
    DO while more #1
        READ #1：s$
        IF rtrim$ (S$[6:20]) <>m$  then
           WRITE #2：s$
        ELSE
            LET no$ =s$[1:5]
            LET name$ =s$[6:20]
            LET price=num(s$[21:28])
            LET quantity=num(s$[29:36]
            LET place$ =s$[37:42]
            PRINT no$ ;“   ”;name$ ;“   ”;
            PRINT price;“   ”;quantity;“   ”;place$
            PRINT “delete this record (y)， not(n)”
            INPUT x$
            IF x$ =“Y” or x$ =“y” then
               PRINT “will delete this record”
               LEt flag=1
            ELSE
                PRINT “do not delete this record”
                WRITE #2：s$
            END IF
        END IF
    LOOP
    IF flag=1 then
       ERASE #1
       SET #2：pointer BEGIN
       SET #1：pointer BEGIN
       DO while more #2
           READ #2：s$
           WRITE #1：s$
       LOOP
    ELSE
        PRINT “Thare is not such recod !”
        PAUSE 10
    END IF
    CLOSE #1
    CLOSE #2
    UNSAVE “stock-d”
END SUB
```

调用该子程序，用户通过新建立的“stock-d”文件，实现对原文件“stock”删除指定的记录。

显示文件记录清单模块

一般地，在一个数据管理系统中都要设置显示记录清单模块。利用该模块可以察看建立或更新操作是否正确。

由于文件中的数据量很大，有时可能多达成千上万条记录，所以显示模块中常利用分页显示技术。下面的子程序即利用这一技术，实现分屏显示库存商品清单功能。

```
! display all records of file
SUB display
    OPEN #1: name "stock", org record
    CLEAR
    PRINT "No. ","name","price","quantity","place"
    LET n=0
    DO while more #1
       READ #1: s$
       LET n=n+1
       LET no$ =s$[1:5]
       LET name$ =s$[6:20]
       LET price=num(s$[21:28])
       LET quantity=num(s$[29:36])
       LET place$ =s$[37:42]
       PRINT no$ , name$ ,price, quantity, place$
       IF end #1 then
          DO
          LOOP until key input
          GET KEY x
          EXIT DO
       END IF
       IF n=20 then
          DO
          LOOP until key input
          GET KEY x
          LET n=0
          CLEAR
          IF end #1 then EXIT DO
          PRINT "No. ","name","price","quantity","place"
       END IF
    LOOP
    CLOSE #1
END SUB
```

调用此子程序，即可把“stock”文件中的全部记录分屏幕显示在屏幕上。每显示一屏之后就自动暂停，待用户察看，然后按任意键，则显示下一屏。在显示最后一屏后按任意键，则返回主程序并显示菜单。

第四节　字节文件

字节文件是 True BASIC 中特有的一种文件类型。从计算机的存储来说，信息的存放，无论是对哪一类文件，都是以字节为单位，将其存储于文件中。True BASIC 提供的字节文件和对字节文件的操作，把磁盘文件视为一个由字节构成的系列（或称字节流），对文件的读出或写入只能以字符串的形式，以字节为单位进行。

字节文件与其他类型数据文件一样，可以打开，关闭，删除，设置文件指针等。但指针位置选择 same，则表示指向上一个字节（即刚操作过的字节）；选择 next，则指向下一个字节；选择 record rn，则指向第 rn 个字节。

字节文件用 READ #语句读出，用 WRITE #语句写入。

字节文件的记录长度可用 READ #语句或用 SET #语句来设定，表示一次读取的字节数，可以根据这个指定的字节数读取任意长度的文件。

一、WRITE #（写字节文件）语句

语句格式：

```
WRITE #n：字符串表达式1，字符串表达式2，……
```

语句的作用是从指针当前位置开始，把字符串表达式结果值中的字符依次写入通道号 n 所对应的字节文件中。

每个字符占一字节，每写一个字符，指针自动向下移动一个字节。

二、READ #（读字节文件）语句

格式1：

```
SET #n：recsize〈字节数〉
READ #n：串变量名1，串变量名2，……
```

格式2：

```
READ #n，bytes〈字节数〉：串变量名1，串变量名2，……
```

语句的作用：从通道号所对应的文件中依次把规定字节数的字符读到各串变量中。

当读完指定的字节数或遇到文件尾标志，则读语句自动停止。

如果在执行读语句的中途，即还未读取规定的字节数，文件就结束了，则将产生运行错误。此时，在程序中可利用 True BASIC 提供的出错处理结构对出现的错误进行处理。

三、字节文件应用举例

建立和使用字节文件一般步骤为：

1. 用 OPEN 语句打开文件

在打开语句中，除了为文件分配一个通道号，用 access，create 短语说明文件的访问方式和建立模式之外，字节文件的组织形式必须用 org byte 说明。

2. 对字节文件进行写入或读出操作

读写操作总是以字符串形式进行。记录大小用于读态，要在读操作前指定记录长度，即一次读取的字节数。

3. 操作结束，关闭文件

〔**例9-23**〕 假如有以下一张学生成绩表：

表9-5

学　　号	性　　别	成　　绩
781201	M	A
781202	W	C
781203	W	B
781204	M	A
781205	M	D
781206	M	E

表中，M 表示男，W 表示女，A，B，C，D，E 分别表示五级记分中的优、良、中、及格和不及格。将上述表中的数据以名为“GRADE”字节文件记在磁盘上。

其程序为：

```
! create a byte file
OPEN #1: name "GRADE",create new, org byte
DO while more data
   READ number$,sex$,grade$
   WRITE #1: number$,sex$,grade$
LOOP
CLOSE #1
DATA 781201,M,A
DATA 781202,W,C
DATA 781203,W,B
DATA 781204,M,A
DATA 781205,M,D
DATA 781206,M,E
END

——————————————————————ex09-23——————————————

OK. run
OK.
```

运行上述程序后，在磁盘上就建立了一个字节文件“GRADE”。

〔例9-24〕 从上面已建立的字节文件“GRADE”中读取数据，每次读取8个字节。

```
! read a byte file
OPEN #1: name “GRADE”, org byte
DO while more #1
   READ #1, bytes 8: x$
   PRINT x$[1:6], x$[7:7],x$[8:8]
LOOP
CLOSE #1
END
```

ex09-24

```
OK. run
781201              M                 A
781202              W                 C
781203              W                 B
781204              M                 A
781205              M                 D
781206              M                 E
OK.
```

利用字节文件的读写语句可以复制任何类型的文件。

〔例9-25〕 将前面建立的记录文件“stock”复制一个副本，取名为“stock-1”。

程序为：

```
! copy a file
OPEN #1: name “stock”,organization byte
OPEN #2: name “stock-1”,create new, org byte
DO
    WHEN exception in
          READ #1, bytes 80: x$
    USE
         LET endflag=1
    END WHEN
    WRITE #2: x$
LOOP until endflag=1
CLOSE #1
CLOSE #2
END
```

ex09-25

```
OK. run
Ok.
```

运行该程序，在磁盘上生成一个名为“stock-1”的新文件，其中的数据及代码形式与原文件“stock”中的完全相同。

这里需要注意的是，打开原文件的语句中，组织形式也要指定为byte，而不管它以前的类型是什么。

程序中的

```
WHEN exception in
        ……
        ……
        ……
USE
       ……
       ……
       ……
END WHEN
```

是True BASIC提供的出错处理结构，它可以屏蔽各种运行错误。出错处理结构的执行情况见第十一章。

利用字节文件不仅可以处理True BASIC状态下建立的各种文件，而且还可以处理在其他软件环境（如wordstar，dbase等）下创建的各种文件，用户不必顾虑所处理的文件是怎样建立的，格式如何。因此，字节文件为用户获取，处理其它软件环境下保存的数据，开辟了一条道路。

习　题

9—1. 简述正文文件，记录文件，字节文件的特点。

9—2. 建立一个名为“ZHIGONG”的正文文件，将若干职工的编号，姓名，性别，出生年月，文化程度，政治面貌写入文件中。

9—3. 把“ZHIGONG”文件中的数据全部读出并显示在屏幕上。

9—4. 将打印机作为一个特殊的正文文件，把“ZHIGONG”文件中的数据在打印机上打印出来。

9—5. 从“ZHIGONG”文件中检索出年龄在30岁以下，文化程度为本科的职工，输出他们的有关数据。

9—6. 在“ZHIGONG”文件中追加若干职工的数据。

9—7. 将若干本书的编号，书名，作者名，定价从键盘输入内存，并用SAVE命令写入磁盘，生成名为“TSGL”的显示格式文件。

9—8. 按提供的作者名，从显示格式文件“TSGL”中检索出有关的图书。

9—9. 写一个通用程序，将任意两个正文文件的内容首尾连接，作为一个新文件写到磁盘上。

9—10. 设正文文件“FILE 1. DAT”和“FILE2. DAT”为按字母已排序的字典文件，要求对它们作归并处理，产生一个包含前两个文件所有数据的有序文件“FILE. DAT”。要求实现归并运算时，不能将“FILE1. DAT”和“FILE2. DAT”中的所有数据调入内存作排序处理。

9—11. 将 N 个学生的学号，姓名及其数学，英语，计算机三门课成绩写入一个多数据项记录文件“score”中。

9—12. 把“score”文件中的所有记录送打印机输出。

9—13. 从“score”文件中检索出三门课总分在240分以上的学生，输出其学号，姓名，各科成绩及总分。

9—14. 编程序实现，从一个正文文件中读出数据，写入一个记录文件中。

9—15. 编程序实现，从一个记录文件中读出数据，写入一个正文文件中。

9—16. 通过字节文件，将“score”文件复制一个副本，名为“score－1”。

第十章　图形程序设计

图常常能以简明的方式提供形象化的数据和变化趋势，使人们增加对复杂现象的了解，并协助作出决策。因此，图形显示技术在计算机辅助设计(Computer Aided Design)、计算机辅助制造(Computer Aided Manufacturing)以及管理事务数据处理等领域得到普及和应用。

True BASIC 具有较强的屏幕图形显示功能。利用作图语句既可以画点、线、方、圆和图形着色，并由此绘制出复杂图形，又可以制作动画，还可以定义图画结构模块，通过调用实现图形的各种变换。

本章将讲述 True BASIC 基本作图语句，并结合管理事务数据处理之需要，介绍一些常用的统计图形的作图算法和程序。

第一节　作图环境

一、屏幕模式

在 True BASIC 初始状态下，屏幕分为编辑窗口和背景窗口。除了这一状态外，True BASIC 还支持其他几种模式。如果程序中使用这些模式，运行程序时编辑窗口和背景窗口将消失，整个屏幕作为完整的窗口显示程序的运行结果或图形。程序结束后按任意键或出现运行错误后，屏幕会自动退出所用模式并返回编辑窗口和背景窗口状态。

计算机在屏幕上显示图形时，系统以象素为单位控制屏幕。象素就是通常所说的屏幕上的“光点”。屏幕象素的密度称为显示器的分辨率。图形的显示质量与显示器的分辨率密切相关，分辨率越高，显示的图形质量就越高。

True BASIC 支持的显示器屏幕模式主要有：

(1) 正文模式(又称文本模式)：用于显示程序的运行结果，既可以是数值型数据，也可以是字符串型数据。整个屏幕可显示25(行)×80(列)个字符，可以同时使用16种颜色。

如果运行含有 CLEAR、光标定位控制语句或者 TAB(row,col)函数的程序，屏幕即由初始状态转换为正文模式状态，运行程序结束后按任意键，屏幕将退出正文模式并返回到初始状态。

(2) 中分辨率图形模式(简称图形模式)：用于显示正文和图形。图形的分辨率为320(水平方向象素数)×200(垂直方向象素数)，文本显示的行列数为25×40。可同时使用四种颜色。

(3) 高分辨率图形模式：用于显示精度较高的图形和正文。分辨率为640×200，文本可显示的行列数为25×80。只能同时显示两种颜色。

以下模式适用于配置 CGA 彩色显示器的计算机系统。如果系统配置的是 EGA 彩色显示器，除上述模式之外还可以使用以下模式：

(4) EGA 高分辨率图形模式：用于显示高精度图形和正文。分辨率为640×350，文本可显示的行列数为25×80。可同时显示16种颜色。

如果系统配置 HERCULES 单色显示器，作图时必须使用以下模式：

(5) HERCULES 模式：用于显示高精度单色图形和正文，分辨率为720×348，正文可显示的行列数为25×80。

屏幕模式是通过使用 SET MODE 语句设置的，其语句格式为：

```
SET MODE〈模式名〉
```

执行该语句时，屏幕的模式将按指定的模式进行设置。其中，模式名为下列字符串(大小写字母不作区分)之一：

“80”	(正文模式)
“Graphics”	(图形模式)
“Hires”	(高分辨率模式)
“Egahics”	(EGA 高分辨率模式)
“Hercules”	(HERCULES 模式)

一个计算机系统能够设置哪些屏幕模式，取决于系统配置的显示器的类型。利用 True BASIC 作图，系统必须配置有彩色监视器和彩色图形适配器。其次，在运行程序过程中，屏幕模式一旦设置就保持其状态，直到被重新设置成另一种模式为止。如果程序中没有设置屏幕模式，则执行到第一个绘图语句时，系统自动将屏幕设置成“Grpahics”(图形)模式。

二、设置图形窗口

利用 True BASIC 作图的一个显著特点是，用户作图时使用的是逻辑坐标，而不是象素坐标，逻辑坐标与象素坐标之间的转换是由系统自动完成的。逻辑坐标的设置是通过使用 SET WINDOW 语句实现的，也就是通过设置图形窗口来建立用户作图的直角坐标系(即逻辑坐标系)。该坐标系一旦建立，一个点的位置便可由该点的坐标唯一地确定，而屏幕则被视为坐标系中的一个矩形区域(即图形窗口)。显示图形时，只显示窗口内的局部图形，超出部分将被“修剪”掉。SET WINDOW 语句格式为：

SET WINDOW 〈X_{min}〉,〈X_{max}〉,〈Y_{min}〉,〈Y_{max}〉

这里，设图形窗口的水平方向为 X 轴，垂直方向为 Y 轴，X_{min}、X_{max}、Y_{min}、Y_{max}均为数值表达式，分别是最小和最大横坐标，最小和最大纵坐标。例如，语句：

SET WINDOW －2,2,－4,4

在屏幕上建立的坐标系如图10-1(a)所示，即坐标原点在屏幕中央，X 坐标的上下限为－2,2,Y 坐标的上下限为－4,4。

又如，语句：

SET WINDOW 1991,1995,0,150

在屏幕上建立的坐标系如图10-1(b)所示。

说明：(1) 设计作图程序时，为了使整个图形位于窗口以内，需要合理地设置窗口坐标，以免部分图形被“修剪”掉。如果程序中未设置图形窗口，当执行到程序中的第一个绘图语句时，系统自动将其坐标设置为0,1,0,1。如果设置的窗口其 $X_{min}>X_{max}$，则绘出的图形将是左右翻转的，如果 $Y_{min}>Y_{max}$，图形将上下颠倒。

(2) 设计通用程序时，可以利用 ASK WINDOW 语句查询当前窗口的坐标，其语句格式

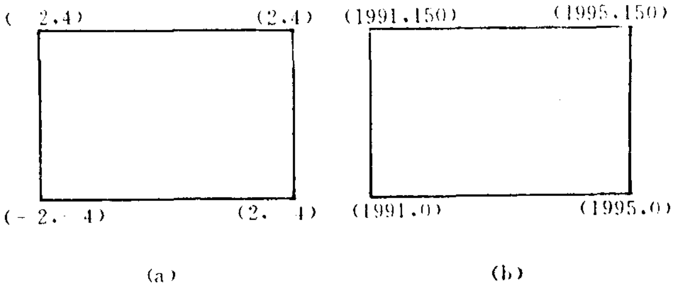


图10-1　窗口坐标的设置示例

为：

ASK WINDOW X_{min}，X_{max}，Y_{min}，Y_{max}

其中，各参数均为数值型变量。

(3) 不同显示器的屏幕纵横比往往不相同。编程时，应当根据显示器屏幕的纵横比调整设置窗口时所用的参数，以便正确反映所绘制图形的形状。例如，大多数 IBM PC 机所配置的显示器，其屏幕纵横比为2∶3.2。如果窗口参数选用不当，就会出现画圆不圆、画方不方。此时，应当重新调整窗口坐标的取值。

〔**例10-1**〕　绘制某省8个地区1993年化肥需求图。设最低需求量为12万吨，最高需求量为55万吨，如果横坐标以地区为单位，纵坐标以万吨为单位，并考虑到图形的可视性，图形窗口可设置为：

SET WINDOW 0,9,10,60

三、图形着色

颜色分为前景颜色和背景颜色。前者指显示的图形或字符的颜色，后者指屏幕没有任何显示时的颜色。作图时，可以使用 SET COLOR 和 SET BACK 语句选择前景色的背景色。

1. SET COLOR(**设置前景颜色**)**语句**

语句格式为：

SET COLOR “颜色名”
或 SET COLOR 〈色号〉

IBM PC 机使用的前景色如表10-1所示。注意，在同一时刻仅能同时使用一组中的三种。

表10-1　　**图形模式下的前景色**

第一组		第二组	
色号	色名	色号	色名
1	green(绿)	4	cyan(青)
2	red(红)	5	magenta(洋红)
3	yellow(黄)	6	white(白)

例如，某图形窗口内已绘制两条曲线，一红一绿，在绘制第三条曲线时，只能置前景色为黄

色。如果置为白色，系统将自动转换为第二组颜色，即第三条曲线画为白色，而第一、二条曲线分别变为洋红色和青色。

语句中，颜色名大小写字母不加区分。若程序中未设置前景色，则系统默认为绿色。

2. SET BACK（**设置背景颜色**）**语句**

语句格式为：

```
    SET BACK "〈色名〉"(或〈色号〉)
或 SET BACKGROUND COLOR "〈色名〉"(或〈色号〉)
```

IBM PC 机使用的背景色如表10-2所示。

表10-2　　**图形模式下的背景色**

色号	色名	色号	色名
0	black（黑）	4	red（红）
1	blue（蓝）	5	magenta（洋红）
2	green（绿）	6	yellow（黄）
3	cyan（青）	7	white（白）

True BASIC 允许在一个 SET COLOR 语句中同时设置前景色和背景色。语句格式为：

```
    SET COLOR "〈前景色名〉"/"〈背景色名〉"
```

3. **查询颜色语句**

（1）查询前景色的 ASK 语句格式为：

```
    ASK COLOR 〈字符串型变量〉
或 ASK COLOR 〈数值型变量〉
```

（2）查询背景色的 ASK 语句格式为：

```
    ASK BACK 〈字符串型变量〉
    ASK BACK 〈数值型变量〉
或 ASK BACKROUND COLOR 〈字符串型变量〉
    ASK BACKROUND COLOR 〈数值型变量〉
```

执行 ASK 语句的结果是将当前颜色的色名（或色号）赋给指定的字符串型变量（或数值型变量）。

此外，True BASIC 还允许查询计算机使用的颜色数目。语句格式为：

```
    ASK MAX COLOR 〈数值型变量〉
```

4. FLOOD（**颜色扩散**）**语句**

语句格式为：

```
    FLOOD x,y
```

该语句功能是，将点(x,y)的颜色扩散到具有相同颜色的邻近的坐标点上。这一语句常被用于闭形区域着色，即先改变该区域内任一坐标点(x,y)的颜色，然后用 FLOOD 语句将此点颜色扩散到与该点原来颜色相同的那些点上。

第二节　基本作图语句

一、PLOT POINTS(画点)语句

语句格式为：

PLOT POINTS：$x_1,y_1;x_2,y_2;\cdots;x_n,y_n$

其中，(x_1,y_1)、(x_2,y_2)、…、(x_n,y_n)表示 n 个点的坐标。如果语句中仅列出一个点的坐标，执行语句时将在指定的坐标位置上显示一个亮点，否则，将显示多个亮点。

如果仅显示一个点时，语句可简写为 PLOT x,y。

〔**例10-2**〕　显示衰减振荡函数 $e^{-0.1x}\cdot\sin x$ 的曲线。

程序为：

```
! Display functional curve
SET WINDOW 0, 6 * pi, -1,1
FOR x=0 to 6 * pi step 0.0625 * pi
    PLOT x,exp(-0.1 * x) * sin(x)
NEXT x
END
```

执行 RUN 命令后，将在屏幕上显示0～6π 的衰减振荡曲线。

二、PLOT LINES(画线)语句

语句格式为：

PLOT〔LINES：〕$x_1,y_1;x_2,y_2;\cdots$〔;〕

执行该语句时，将相邻坐标点以直线连接起来。语句末尾有分号时，表示末点坐标将与下一个画线语句的第一个坐标点以直线相连接；无分号时，则表示不连接。

与 PRINT 语句相仿，该语句也可以只有“PLOT”，其功能是切断前一画线语句带分号的末坐标点与后一画线语句第一坐标点之间的连线。

〔**例10-3**〕　在同一屏幕上绘制某地区自1985年到1993年彩色电视机普及率和供应量趋势图。

分析：横坐标表示时间，即1985年到1993年，纵坐标表示普及率和供应量。由于普及率和供应量的取值范围不同，因此需要设置两次窗口坐标。绘制普及率折线时，图形窗口设置为：SET WINDOW 1985,1993,15,41，并将折线颜色设置为红色；绘制供应量折线时，图形窗口设置为：SET WINDOW 1985,1993,17,54。鉴于区分两条折线，将供应量折线设置为绿色。

程序为：

```
! Draw two trend-graphs
SET Window 1985,1993,15,41
SET color "red"
FOR year=1985 to 1993
```

```
          READ popul
          PLOT year,popul;
NEXT year
PLOT
SET window 1985,1993,17,54
SET color "green"
FOR year=1985 to 1993
          READ val_supply
          PLOT year,val_supply;
NEXT year
PLOT
! Data block:rate of popularity
DATA 15.6,17,18.6,20.9,23.5,27.8,32.7,36.8,40.9
! Data block: value of supply
DATA 17.68,21.43,24.71,30.38,33.33,39.17
DATA 46.71,53.61,53.13
END
```

由程序可以看出，绘制趋势图时，图形窗口各参数的设置取决于数据的取值范围。因此，编写绘制趋势图通用程序时，必须设计一个逻辑模块，用于计算一组数据的最大值和最小值。分别调用该模块求出两组数据的最大值和最小值，并将它们作适当的修正（比如，增加或减少10%后取整）即可作为设置图形窗口的参数值。请读者编写出绘制趋势图的通用子程序。

三、PLOT ARER(区域着色)语句

语句格式为：

PLOT AREA：$x_1,y_1;x_2,y_2;\cdots;x_n,y_n$

执行该语句时，先从点(x_1,y_1)到点(x_2,y_2)画一条线，再从点(x_2,y_2)到点(x_3,y_3)画一条线，…，最后再从点(x_n,y_n)到点(x_1,y_1)画一条线，即画一条由 n 个坐标点组成的封闭曲线，然后用当前颜色对该区域着色。

〔**例10-4**〕 绘制六个绿红黄三色相间的三角形。

分析：每绘制一个三角形均需用 SET COLOR 和 PLOT AREA 语句设置颜色和画三角形。然而，由于题目要求六个三角形的颜色依次为绿红黄绿红黄，即颜色的色号分别为1,2,3,1,2,3，因此，可以将颜色的色号描述为“mod(int(x/3),3)+1”，其中 x 分别取0,3,6,…,15。

程序为：

```
! Draw triangles
SET window 0,18,0,12
OPTION ANGLE degrees
FOR x=0 to 15 step 3
          SET color mod(INT(x/3,3)+1
    PLOT AREA: x,0;(x+1.5),1.5*tan(60);x+3,0
NEXT x
END
```

四、画矩形和实心矩形

用 PLOT LINES 语句可以画矩形，但绘图速度较慢。True BASIC 中的 BOX LINES 和 BOS AREA 语句可分别实现快速画矩形和实心矩形。其语句格式为：

BOX LINES $X_{min}, X_{max}, Y_{min}, y_{max}$
BOX AREA $X_{min}, x_{max}, Y_{min}, Y_{max}$

其中，参数的设置与设置图形窗口时所用的参数意义相同。执行这两个语句都将在屏幕上快速绘制矩形，其区别在于后一语句还要以当前颜色填充矩形，即绘制实心矩形。

〔**例10-5**〕 画四个大小各异颜色相间的正方形。

程序为：

```
! Draw four squares
SET window -14,14,-10,10
FOR i=1 to 4
    SET color mod(i,3)+1
    BOX LINES -2*i,2*i,-2*i,2*i
NEXT i
END
```

五、画圆和实心圆

BOX CIRCLE（画圆）语句格式为：

BOX CIRCLE $X_{min}, X_{max}, Y_{min}, Y_{max}$

功能：在参数给定的正方形框内，绘出其内接圆。如果参数给定的是矩形，则绘出椭圆。

如果要绘实心圆，则在 BOX CIRCLE 语句后增加一个 FLOOD 语句，该语句中的坐标点(x,y)应是圆内任一点。

〔**例10-6**〕 绘制图形比例图。

在管理事务数据处理中，常用圆形比例图表示某一整体的各个部分所占比例。该图亦称作扇形图，或饼形图。它是将圆形按比例分割成若干个扇形，每个扇形表示某部分所占比例。

下面的程序将绘制一幅圆形比例图，以表示某院校专职教师的职称结构。职称分为教授(professor)、副教授(associate professor)、讲师(lecturer)、助教(assistant)四类。

```
! Draw circular proprotional graph
DIM title(4),prop(4),angle(4)
MAT READ title
LET total=0
FOR i=1 to 4
    LET total = total + title(i)
```

```
NEXT i
MAT prop=(round(100/total,3)) * title
MAT angl=(round(360/100,3)) * prop
OPTION ANGLE degrees
SET window -2,2,-1.5,1.5
BOX CIRCLE -1,1,-1,1
SET back "blue"
LET alpha, sum = 0
ROR i=1 to 4
    IF mod(i,2) = 0 then
       SET color "red"
    ELSE
        SET color "green"
    END IF
    LET sum=sum+angl(i)
    FOR r=alpha to sum-1
        PLOT AREA: 0,0; cos(r),sin(r);cos(r+1),sin(r+1)
    NEXT r
    LET alpha = alpha + angl(i)
NEXT i
SET cursor 2,1
PRINT using "professor: ##.#":prop(1)
PRINT using "ass-professor: ##.#":prop(2)
PRINT using "lecturer: ##.#":prop(3)
PRINT using "assistant: ##.#":prop(4)
DATA 6,55,105,159
END
```

程序中，title 数组用于存放四种职称的教师人数，prop 数组用于存放各职称人数占全体人数的百分比，angl 数组用于存放圆形比例图中各扇形的夹角（单位为度）。数组 prop 和 angl 各元素的值是通过 MAT 赋值语句获得的。

圆形比例图的绘制利用双重 FOR 循环结构实现，其中 r 循环实现每个扇形的绘制。

运行此程序，将绘制由四个扇形组成且红绿颜色相间的比例图，图的左侧标出各类职称所占的百分数。

六、图形中的正文设置

绘图时，如果将必要的文字或符号与图一并显示，例如标题、坐标单位、刻度值等，可使图形更明了、更实用。例10-6程序利用 SET CURSOR 语句和 PRINT USING 语句将必要的正文信息与图同时显示在屏幕上。除了这一方法外，True BASIC 还允许使用 PLOT TEXT 语句在图形中设置正文。该语句格式为：

```
    PLOT TEXT,AT x,y:〈字符串表达式〉
或 PLOT TEXT,AT,x,y:USING $ ("〈格式串〉",〈输出项〉)
```

执行该语句时，将把字符串表达式的值显示在坐标点(x,y)的右上方，或者将输出项的值按格式串中指定的格式予以显示。其中格式串的用法与 PRINT USING 语句中的用法相同。

〔例10-7〕 绘制直方图示例。

直方图是用矩形的高度反映相应的数量值，由各矩形的高低来直接描述某些事物之间数值大小的对比关系。因此，它被广泛应用于管理事务数据处理领域。例如，企业一年内各季度或1至12月份完成产品产量、产值的增减状况对比等等。

与绘制趋势图相仿，绘制直方图时也需要根据数据的取值范围确定窗口坐标值。下面将某厂各季度完成的产值绘制成直方图。

程序为：

```
! Draw histogram
SET window 0,10,0,120,
PLOT LINES: 0.5,10;9,10
FOR x=1 to 7 step 2
      READ value_p
      BOS AREA x, x+0.5,10,value_p
      PLOT TEXT,at x, value_p+2: using $ ("##.#",value_p)
      PLOT TEXT,at x-0.5,1: using $ ("#quarter",(x+1)/2)
NEXT x
SET cursor 1,10
PRINT "production histogram"
DATA 82.6, 89.1,85.7,94.8
END
```

程序中，PLOT LINES 语句用于绘制横轴，PLOT TEXT 语句在图中标出相应的文字说明信息。

七、MAT PLOT 簇语句

用 PLOT POINTS、PLOT LINES、PLOT AREA 语句绘图时，如果坐标点很多，书写起来就很不方便。为此，True BASIC 提供了三个利用数组进行绘图的语句。语句格式为：

```
MAT  PLOT  POINTS:〈数组名〉
MAT  PLOT  LINES:〈数组名〉
MAT  PLOT  AREA:〈数组名〉
```

上述语句分别实现按数组提供的坐标点画点、线，或者画一个封闭曲线并按当前色着色。但是在使用这簇语句作图时，数组必须是二维的，并且每一行的第一列元素应存放 x 的坐标值，第二列元素存放 y 的坐标值。

读者可以修改例10-3程序，将数据组织成两个二维数组，然后利用 MAT PLOT LINES 语句在同一屏幕上绘制两个趋势图。

第三节 动画和图画结构

一、动 画

True BASIC 实现动画的方法是，先把图形绘好，并把图存起来，然后反复执行抹图和在新的位置再现。由于这一过程快速进行，因此产生动画效果。与制作动画有关的语句是：BOX KEEP（存图）语句、BOX CLEAR（抹图）语句和 BOS SHOW（再现）语句。

BOS KEEP 语句格式为：

BOX KEET X_{min}，X_{max}，Y_{min}，Y_{max} IN〈字符串变量〉

其中，各参数是包含已绘图形的矩形框的边界。执行该语句时，将矩形框内的图形以及前景色和背景色一并存储在字符串变量中。

BOX CLEAR 语句格式为：

BOX CLEAR X_{min}，X_{max}，Y_{min}，Y_{max}

功能：将参数指定的矩形框内的图形抹去。

BOX SHOW 语句格式为：

BOX SHOW 〈字符串变量〉AT X_{min}，Y_{min}

功能：以点（X_{min}，Y_{min}）为左下角坐标，将存储于字符串变量中的图形及颜色信息，再次显示出来。

〔**例10-8**〕 模拟乒乓球在盒内跳动。

程序为：

```
! Play ping-pang
SET window 0,30,0,20
SET color “white”
SET back “red”
LET c = 0.5
LET d = 1
LET x = 15
LET y = 12.5
BOX CIRCLE x, x+1,y, y+1
FLOOD x+0.5,y+0.5
BOX KEEP x, x+1,y,y+1 in ball$
PLOT TEXT, at 4,1:“press any key to end!”
DO
    BOX CLEAR x,x+2,y,y+2
    LET x = x+c
    LET y = y+d
```

```
        IF x 〈0.5 or x〉 28.2 then LET c = —c
        IF y 〈5.5 or y〉 18.5 then LET d = —d
        BOX SHOW ball$ at x,y
        BOX LINES 0,30,5,20
    LOOP until key input
    END
```

细心的读者会发现，BOX CLEAR 语句中指定的矩形框比存图时的矩形框稍大了一些，作此修正的目的是将矩形边缘上的点也一同抹掉，否则会遗留一些痕迹。

二、图画结构

1. 图画结构的定义与调用

为实现模块化程序设计，True BASIC 提供了一种类似于子程序的图画结构，通过调用可以实现图形的放大、缩小等各种变换。图画结构的定义格式为：

```
    PICTURE 〈图画名〉(〈参数1〉[,〈参数2〉]…)
        ……┐
        ……├  语句序列
        ……┘
    END PICTURE
```

定义图画结构与定义子程序相仿，不同之处在于：图画结构以 PICTURE 开始，以 END PICTURE 结束，并且语句序列中至少有一个绘图语句。其他两者完全相同，不再赘述。

定义图画结构的目的在于构造作图逻辑模块，以及实现图形的几何变换。图画结构的调用使用 DRAW 语句，其格式为：

```
    DRAW 〈图画名〉(〈参数1〉,〈参数2〉,…)
    DRAW 〈图画名〉(〈参数1〉,〈参数2〉,…) WHIT 〈变换〉
    DRAW 〈图画名〉(〈参数1〉,〈参数2〉,…) WHIT 〈变换1〉*〈变换2〉*…
```

其中，实在参数表是可选项，〈变换1〉*〈变换2〉*…为复合变换，各变换之间以星号“*”分隔，进行变换时按自左至右顺序。

图画结构既可以嵌套调用，也可以调用自身，即递归调用。

2. 图画的几何变换

True BASIC 提供有以下五种基本变换，用户可以利用 DRAW 语句将其进行组合以构成更为复杂的变换。

SHIFT(a,b)　表示将图画中的所有点(x,y)位移到(x+a,y+b)，即平移变换。

SCALE(a)　表示将图画中的所有点(x,y)变为(x*a,y*a)，即缩放变换。

SCALE(a,b)　表示将图画中的所有点(x,y)变为(x*a,y*b)，即改变图形的横纵比。

ROTATE(a)　表示绕窗口坐标原点逆时针方向将图画旋转 a 角度或弧度，即旋转变换。

SHEAR(a)　表示将图中所有垂线顺时针倾斜 a 角度或弧度，即切变变换。

有兴趣的读者可以利用前述画线语句构造一个绘制小房子的图画结构，然后在主程序中对小房子作各种变换处理。

注意，如果在图画结构中使用了 BOX 簇作图语句，调用该图画时不能作任何几何变换，原

因是BOX“盒子”的位置是固定的。

第四节　图形输入和多窗口操作

一、GET　POINT(图形输入)语句

利用GET　POINT语句和GET　MOUSE语句可以实现图形坐标点的输入。后一语句适用于配置有鼠标器的计算机系统。本节仅介绍GET　POINT语句，其语句格式为：

```
GET  POINT:x,y
```

执行该语句时，屏幕上首先出现一个形状为“＋”的作图光标，它表示光标当前所处的位置。用户可以使用光标移动键(←、→、↑、↓、Tab、PgUp、PgDn、End、Home等)将光标“＋”移到需要位置，按回车键后，系统就把光标所处位置的坐标值赋给变量x和y。

二、多窗口操作

True BASIC既允许把整个屏幕作为一个图形窗口，也允许用户将屏幕划分成几个图形窗口，实现多幅图形或正文在同一屏幕上显示。

1. 打开图形窗口

打开图形窗口使用OPEN　SCREEN语句，其格式为：

OPEN　#n: SCREEN X_{min},X_{max},Y_{min},Y_{max}

其中，#n为通道号，使用规则与文件通道相同。由于该语句把整个屏幕窗口的坐标视为0,1,0,1，因此，语句所列参数必须在0到1之间取值。

执行该语句时，将在屏幕上打开一个新窗口，其位置和大小取决于语句中的参数值。例如，要在屏幕右下角建立一个新窗口，则应使用以下语句打开窗口：

OPEN　#2: SCREEN 0.6,1,0,0.4

开辟的新窗口如图10-2所示。

为了满足作图要求，可以在打开窗口后重新用SET　WINDOW语句设置窗口坐标。

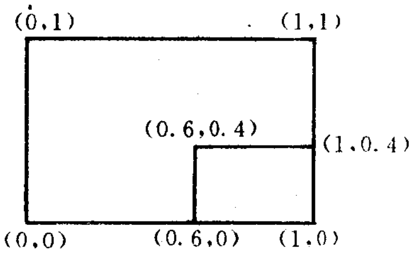


图10-2　打开的一个新窗口

2. 窗口之间的转换

程序中可以同时打开几个图形窗口，但同一时刻，只能在一个窗口内绘图或显示正文，该窗口称为当前窗口。若程序中仅打开一个窗口，则这个窗口就是当前窗口。若程序中同时打开了几个窗口，则最后打开的那个窗口为当前窗口。如果要在非当前窗口内绘图或显示正文，则必须将其设置为当前窗口。

设置当前窗口是使用语句

WINDOW　#n

实现的，其中#n的通道号。执行该语句时，除了转换窗口外，系统还将把原窗口的窗口坐标、当前颜色、光标位置等信息存储起来，以利于以后恢复。

设置当前窗口后，所有绘图语句以及PRINT、INPUT和光标设置语句都将在这一窗口中

进行。如果设置光标位置时，应在当前窗口所允许的范围内确定光标的行列值，否则，系统将给出错误信息。利用 ASK MAX CURSOR 语句可以查询当前窗口行和列的取值范围。

3. 窗口的关闭

当一个窗口内的操作结束时，应使用“CLOSE ＃n”语句将其关闭。执行该语句后，True BASIC 就自动转换到＃o 窗口，即整个屏幕。如果需要改为其他窗口，仍需重新设置当前窗口，或打开一个新的窗口。

〔**例10-9**〕 建立两个窗口，左边的窗口(＃1)用来输出图形，右边的窗口(＃2)用来显示文本信息。

程序为：

```
! Using two windows
OPEN #1: screen 0, 0.49,0.1
SET window 0,10,0,120
SET color "yellow"
SET back "green"
PLOT LINES: 0.5,5;9,5
FOR x = 1 to 7 step 2
      READ value_p
      BOX AREA x, x+1,5.5,value_p
      PLOT TEXT,at x, value_p+1:using $ ("##.#",value_p)
NEXT x
OPEN #2: screen 0.5,1,0,1
SET color "red"
SET back "green"
SET cursor 10,1
PRINT "Production histogram"
SET cursor 11,1
PRINT repeat $ ("-",20)
RESTORE
FOR x=1 to 4
      READ value_p
      SET cursor x + 11, 4
      PRINT using "# quarter####.#";x,value_p
NEXT x
DATA 82.6,89.1,85.7,94.8
END
```

执行 RUN 命令后，整个屏幕被划分成左右两个窗口。左侧窗口绘制全年各季度完成产值的直方图，其前景色为黄色，背景色为绿色。右侧窗口则显示出与直方图有关的正文信息，其前景色为红色，背景色为绿色。

习　题

10—1. 在同一坐标系中，绘制函数 $y=a^x$ 的图形，其中 a 分别取0.5，2，3，并用不同颜色对函数曲线着色。

10—2. 画一个矩形，其左下角坐标为(0.5，0.5)，右下角坐标为(2，3)，并绘出它的两条对角线。

10—3. 画一个正方形，然后画两条线把它分割成四个大小相等的小正方形，并对其着色。注：如果绘出的正方形不方，应调整窗口横纵比。

10—4. 某批发站胶雨衣批发量历史数据如表10-3所示：

表10-3　**胶雨衣批发量**　(单位：百套件)

季度 数量 时间	一季度	二季度	三季度	四季度
1991年	5562	5061	4776	3237
1992年	5970	6428	4943	3582

试编制程序，在屏幕上绘制两条趋势图，并用不同颜色着色。

10—5. 某市东明商场1992年1～6月销售额分别为210万元，182万元，173万元，167万元，194万元，191万元。试编写程序绘制销售统计直方图。

10—6. 某工厂生产第一线工人年龄结构为：

18～25岁：465人

26～35岁：392人

36～50岁：178人

51～55岁：49人

试编写程序，绘制年龄结构圆形比例图。

10—7. 编写一个绘制趋势图的通用子程序 trend(value(，))，其中 value(，)为二维数组，每行第一列元素为横向坐标值，每行第二列元素为纵向坐标值。

10—8. 编写一个绘制扇形图的通用子程序 fan_shaped(value())，其中 value()为一维数组，各元素为一整体各部分的取值。

10—9. 编写通用子程序 histogram(value()，text_m $())，实现在一个窗口内根据数组参数 value()各元素的值绘制直方图，在另一窗口内显示数组参数 text_m $()中的正文信息。

第十一章　程序的调试与出错处理

程序设计过程中，出错是难免的，尤其是初学者。因此，发现与消除程序中的错误，即程序调试，是程序设计中必不可少的一个重要环节。本章首先讨论程序中常见错误的类型及其调试技术，然后介绍 True BASIC 特有的抑制运行态错误的出错处理结构，以及与此有关的函数和语句。

第一节　程序中的常见错误类型及调试

一、程序中的常见错误类型

不同人员编制不同的程序，出现的错误可能千差万别，但归纳起来，常见的错误有以下四类。

1. 语法错误

语法错误是指程序中出现了违反有关语法规则的错误，由于这类错误致使编译过程无法通过，因此又称为编译态错误。例如，程序设计人员对所用的计算机语言不熟练或者粗心，写错了语句的关键字，使用了非法的变量名或函数名，语句格式不符合规定，结构性语句中出现结构缺陷，使用数组未预先定义等等，都会造成语法错误。调试程序时，首先要找出并消除程序中的全部语法错误。方法是，将 True BASIc 程序输入计算机后，发 RUN 命令，系统进入解释状态，对程序边解释边执行。一旦发现某个语句有语法错误，光标则停在编辑窗内有错误的语句处，并在背景窗内显示出相应的错误信息。程序设计人员可根据附录 B 中对错误信息的解释，对出错语句进行分析，找出错误并予以改正。然后使光标移到背景窗，再发 RUN 命令，如还有语法错误，光标便停于编辑窗内下一个有错误的语句行。依此类推，直到所有的语法错误被消除。

2. 逻辑错误

所谓逻辑错误是指编程人员自身设计逻辑产生的错误。这类错误的具体表示形式是程序能够运行，但输出的结果却不正确。

导致程序产生逻辑错误的主要原因有：

为变量提供初始值错；

逻辑判断条件与题意不符；

表达式写错；

算法本身存在错误。

由于系统不会显示逻辑错误，所以查找程序中的逻辑错误是比较困难的。有些较大或较复杂的程序，在使用很久之后，还发现有逻辑错误。

3. 计算错误

这里所说的计算错误是程序能够运行，但受计算机系统自身的限制，造成计算结果出错。

常见的计算错误主要有：

（1）为变量提供的数据，超出了系统可以表示的范围而引起的错误。

（2）因存储单元无法精确表示一般的十进制小数而产生误差，误差的积累导致的计算错误或条件判断错误。

计算错误除了引起输出结果可能出错外，有时还会影响循环结构的循环执行次数，甚至出现“死循环”。

下面是由于计算错误造成条件判断错，使循环失控的一个例子。

〔**例11-1**〕

```
let x=9.99
do
    let x=x+0.001
    print x
    if x=10 then exit do
loop
end
```

运行程序时发现陷入了“死循环”，说明 x 值永远不等于10，即使屏幕上输出 x 为10时，也不退出循环。改正的办法是在程序中尽量避免使用两个数值量是否相等来作为控制条件。本例中，可将 x=10改为 x>=10即可。

由于系统也不会显示出程序中何处出现了计算错误，必须由程序设计人员分析查找，并予以消除。

4. 运行态错误

如果一个程序中各语句均不存在语法错误，但程序却无法顺利执行，则表明程序存在运行态错误。常见的运行态错误主要由误操作引起。例如，读写磁盘文件时，插错磁盘或忘了关上驱动器门；打印输出时，未接通打印机或没有装好打印纸；键盘输入数据时，数据与接收变量的类型不匹配，或造成除数为零，对负数开平方等。反映存在运行态错误的现象是程序执行的被中断，必须消除引起运行态错误的原因或屏蔽产生的运行态错误，程序才能继续执行。

二、程序调试技术

程序经过排除语法错误之后，仍滞留于程序中的逻辑错误和计算错误通常是隐蔽的。为了发现和纠正这些错误，必须采用有效的方法和策略，即程序调试技术，以便迅速捕获与这些错误有关的信息。但需要强调的是，对程序进行调试的目的是发现错误，纠正错误，而无法证明程序没有错误。

常用的调试技术有：追踪，中断，强制和借助于调试工具。

1. 追踪技术

追踪的目的是检查程序是否按预定流程运行。多用于检查条件测试、循环结构、计算公式、调用逻辑模块时的参数传递等是否正确无误。实现追踪有以下三种方式：

（1）追踪输出

为了提供程序执行的路径，以便从中找到线索，在程序中设置一些 PRINT 语句，设置的位置通常是：

逻辑模块的首部或尾部；

调用语句前后；

循环结构内的第一个语句或最后一个语句；

循环结构后的第一个语句；

分支点之前；

分支中的第一个语句；

表达式计算赋值后的第一个语句。

(2) 设置计数器变量

可以在每个逻辑模块或基本结构中，设置一个计数器变量，执行程序时，每进入该结构一次·便计数一次。这样不仅可以判断程序的流程，而且当程序存在死循环时，用此方法能很快确定。

(3) 设置状态标志变量

在每个逻辑模块中设置一个状态标志变量，程序进入该模块时，便给该变量一个特殊值，根据标志变量的值，可以判断程序流程的大致路径。

采用追踪方式调试程序时，可采取先疏后密的策略设置 PRINT 语句，即先把错误隔离在一个区域，再在该区域内设置 PRINT 语句，将含错区域逐步缩小。

2. 中断技术

中断技术是指在程序中设置断点。这一技术常用于检查循环结构，逻辑模块内某些变量的变化。

在 True BASIC 中，设置断点的作法是：当需要在某行设置断点时，先用 F4键将该行做标记，然后在背景窗键入 BREAK 命令，这时，在标记行的行首出现“>”号，表示设置的断点位置。程序运行到断点时暂停，在背景窗用 PRINT 命令可将有关变量的值显示出来。

若要自断点处继续往下运行程序，可用 CONTINUE 命令实现。若要取消断点时，只须再次键入 BREAK 命令即可。

3. 强制技术

强制技术就是利用调试用例，迫使程序逐个通过所有可能出现的执行路径，系统地排除无错的程序分支，逐步缩小检查的范围。

利用这一技术的关键在于设计调试用例。

4. 借助调试工具

为了使用户方便地调试程序，True BASIC 2.03版提供了以下调试工具：

TRACE·TRU　　TRACE2·TRC　　TRACE3·TRC

其中，后两个是 True BASIc 的库文件。

调试命令的一般格式为：

```
DO TRACE option(var 1, var2, …)
```

其中 var1，var2，…是欲追踪的变量名，一次允许同时追踪多达8个变量，但只允许简单变量；option 可以是“step”，“Slow”，“fast”，“break”四种选择之一。

Step 表示单步追踪，按一次空格键执行一个语句；Slow 表法慢速追踪；fast 表示快速追踪；break 表示开始时快速追踪，当满足给定条件时，转为单步追踪。

具体使用时，option不论是选择哪一种，在跟踪时均可以利用按键随意改变追踪速度：按空格键变为单步追踪；按“s”键变为慢速追踪；按“f”键变为快速追踪。

执行DO TRACE命令时，屏幕的右侧出现两个窗口，被追踪的变量名及其当前值显示在右上窗口，程序的运行结果显示在右下窗口，屏幕的左侧则显示被追踪的程序段。追踪开始时，有一条光带罩在在第一条将要执行的语句上，此后，随着程序的追踪执行，光带总是罩在下一条将要执行的语句上。

第二节　出错处理结构

为避免程序执行时因运行态错误面发生“致命”中断。True BASIC提供了一种称为陷井技术的处理运行态错误的手段——出错处理结构。利用这一结构，可以屏蔽各种运行态错误，提高程序的坚固性。

出错处理结构的格式为：

```
WHEN ERROR IN
        ……
        ……  } 被保护模块
        ……
USE
        ……
        ……  } 出错处理模块
        ……
END WHEN
```

在执行出错处理结构时，首先执行WHEN与USE之间的被保护模块，若无运行态错误发生，则执行被保护模块后，跳过出错处理模块，执行END WHEN的后继语句；如执行被保护模块期间出现运行错误，便停止执行此模块中的剩余部分，立即转向相应的出错处理模块，执行其中的语句，实现出错处理，然后再执行END WHEN的后继语句。

由于将程序中可能产生运行错误的程序段设置为出错处理结构，可以使程序在出现运行错误时自动进入出错处理模块，错误经处理后，程序得以继续往下执行。因此在开发实用程序过程中，这种方法常被广泛采用。

〔**例11-2**〕　从数据文件中读取数据时，有时会因插错了磁盘找不到指定文件，或未将驱动器小门关闭而导致出现运行态错误。下面的程序是读取一个正文文件中的数据，并为可能发生的误操作设置了出错陷井。

程序为：

```
! read a text file
DO
     LET flag=0
     WHEN erroe in
             OPEN #1: name "data-1",organization text
             DO while more #1
```

```
            LINE INPUT #1: x$
            PRINT x$
        LOOP
        LET flag=1
    USE
        PRINT “no such file!”
        PRINT “insert another disk!”
        PRINT “close door and press any key !”
        DO
        LOOP until key input
        GET KEY x
    END WHEN
LOOP until flag=1
END
```

运行此程序时，如不发生误操作，则顺序读出并显示指定正文文件中的全部数据，出错处理模块将不被执行。如果发生误操作，则立即进入出错处理模块，显示提示信息，然后等待用户按任一键，在这期间，用户可采取措施，为读文件作好各项准备工作。

几点补充说明：

（1）除函数，子程序，图画结构的定义之外，被保护模块中可以出现其他任何语句，包括对函数，子程序或图画的调用语句。

（2）被保护模块中，允许 WHEN-END WHEN 结构的嵌套使用，并且可以使用 EXIT HANDLER 语句从嵌套的出错处理模块中退出。

（3）在被保护模块中若调用函数，子程序或图画，那么它们的所有语句均被调用层的出错处理模块所保护。在这些逻辑模块的定义中也可以有它们自己的出错处理模块。

（4）在出错处理模块中，不允许有任何保护模块。

（5）关键字 ERROR 可用 EXCEPTION 代替。

第三节　与出错处理有关的函数和语句

一、出错函数

为了便于用户编写出出错处理模块，True BASIC 提供了以下几个出错函数。

1. EXTYPE 函数

该函数的功能是返回当前发生错误所对应的代码（出错号）。

出错号不同，表示错误的性质不同，规定如下：

0	无错误发生
1～999	供用户选用
1000以上	由 True BASIC 提供出错号。

系统提供的出错信息及相应出错号见附录 B。

2. EXTEXT $ 函数

该函数返回一个字符串，其作用是用文字注释当前发生了什么错误，如果未发生运行错误，此函数返回一个空串。

3. EXLINE $ 函数

此函数返回一个字符串。有错误发生时，此串表示程序中发生错误的位置，即给出出错的行号及程序名或子程名或自定义函数名。系统规定主程序名为“Main program”。无错误发生时，返回空串。

行号是从整个程序开头处(而不是从某个自定义函数或子程序开头处)按顺序计算的，第一行为1，第二行为2，依此类推。一个注释行或一个空行也要算作一行。库文件中出错所给出的行号，同样是对整个库文件而言的。

〔**例11-3**〕 出错函数应用举例

```
! suing extype,extext $  and exline $
DEF badfunc(x)=sqr(x)/x
SUB badsub(y,z)
     LET s=1/y+1/z
     PRINT s
END SUB

! main program
WHEN error in
        INPUT prompt “Enter a,b,c:”: a,b,c
        PRINT badfunc(a)
        CALL badsub(b,c)
USE
       PRINT “code of error:”
       PRINT extype
       PRINT “text of error:”
       PRINT extext $
       PRINT “place of error:”
       PRINT exline $
END WHEN
END
```

——————————————————ex11-3——————————————

```
OK. run
Enter a,b,c:4,5,6
  .5
  .366667
Ok. run
```

```
Enter a,b,c:-5,-4,-6
code of error:
  3005
text of error:
SQR of negative number.
place of error:
2 in badfunc in Main pgrogram,11 in Main program
Ok. run
Enter a,b,c:7,8,0
  .377964
code of error:
  3001
text of error:
Division by zero.
place of error:
4 in badsub in Main program,12 in Main program
Ok.
```

上述程序中定义了一个自定义函数和一个无参子程序，主程序中使用了出错处理结构。通过阅读程序和分析相应的运行结果，读者可进一步理解三个出错函数的作用。

二、EXIT HANDLER(退出错误处理模块)语句

当一个出错处理模块不能恰当地处理某一运行错误时，可以利用 EXIT HANDLER 语句从一个出错处理模块中退出，返回到其外层出错处理模块，以便对该错误进行处理。如果没有外层出错处理模块时，则返回 True BASIC 系统，并显示出错信息。EXIT HANDLER 语句只能用于出错处理模块中，并且通常用于嵌套或递归调用的出错处理。

该语句格式为：

```
EXIT HANDLER
```

〔例11-4〕 EXIT HANDLE 语句应用举例

计算 LOG(x)/SQR(y)的值，可能出现三种运行态错误(当 $x \leqslant 0$时，取对数出错；当 $y=0$时，产生除数为0的错；当 $y<0$时，开平方出错)。为处理计算表达式时可能产生的这些运行态错误，程序采用了嵌套出错处理结构和退出错误处理模块语句。

程序如下：

```
! demonstrate exit handler sentence
INPUT prompt "Enter x,y:":x,y
WHEN error in
         WHEN error in
                  WHEN error in
                           LET result=log(x)/sqr(y)
```

```
                USE
                        IF extype <>3004 then EXIT HANDLER
                        PRINT "x<=0,log of number <=0"
                END WHEN
        USE
                IF extype <>3001 then EXIT HANDLER
                PRINT "y=0,division by zero"
        END WHEN
USE
        IF extype=3005 then
           PRINT "y<0,sqr of negative number"
        END IF
ENE WHEN
PRINT "result=";result
END
```

ex11-4

```
OK. run
Enter x,y:6,8
result=   .633483
Ok. run
Enter x,y:-4,5
x<=0,log of number<=0
result =   0
Ok. run
Enter x,y:8,0
y=0,division by zero
result =   0
ok. run
Enter x,y:4,-6
y<0,sqr of negative number
result=   0
Ok.
```

该程序使用了三层出错处理结构，运行时，当出错号(即函数 extype 的值)为3004时，显示出“x<=0，Log of number <=0”的错误信息，否则通过执行第三层中的退出错误处理模块语句，转入第二层错误处理模块。此时，若检测到出错号为3001时，则显示出“y=0，division by zero”的错误信息，否则通过第二层中的退出错误处理模块语句转入到第一层出错处理模块中。在这里，若检测到出错号为3005时，则显示出“y<0，sqr of negative number”。

值得指出的是，当输入的 x，y 值，使计算表达式 Log(x)/Sqr(y)产生运行态错误时，对数

值型变量 result 的赋值未能进行，该变量中保留初始化值零。

三、CAUSE ERROR(**捕错**)语句

使用系统提供的 EXTYPE 和 EXTEXT $ 函数，可获得当前发生错误的代码及错误信息。除此以外，用户还可使用 True BASIC 提供的 CAUSE ERROR 语句自行定义出错代码和出错信息，以扩大捕获错误的范围。

该语句格式为：

```
CAUSE ERROR 〈数值表达式〉〔,〈字符串表达式〉〕
```

其中数值表达式为用户定义的出错代码，通常取1～999之间的任意整数，如果带有字符串表达式，则提供出错的文字注释。

CAUSE ERROR 语句既可以用在出错处理结构之中，也可以用在出错处理结构之外。如用在出错处理结构中，它捕捉到的错误可由错误处理模块所处理；若在其外，且语句中带有字符串表达式时，True BASIC 系统便将错误信息显示出来。

〔**例11-5**〕 CAUSE ERROR 语句应用举例

```
! demonstrate cause error sentence
RANDOMIZE
DO
    LET a=10+int(rnd * (100-10))
    LET b=10+int(rnd * (100-10))
    INPUT prompt “Enter operation code:”: code $
    WHEN error in
          SELECT CASE code $
          CASE “+”
               PRINT a;“+”;b;“=”;a+b
          CASE “-”
               PRINT a;“-”;b;“=”;a-b
          CASE “ * ”
               PRINT a; “ * ”;b;“=”;a * b
          CASE “/”
               PRINT a;“/”;b;“=”;a/b
          CASE else
               CAUSE ERROR 555,“error of operation code”
          END SELECT
    USE
         PRINT “code of error:”;extype
         PRINT extext $
    END WHEN
    INPUT prompt “continue ?(y/n)”:x $
```

```
LOOP until x $ =“N” or x $ =“n”
END
```

ex11-5

```
Ok. run
Enter operation code:+
41+13=   54
continue ? (y/n)y
Enter operation code:-
92-19=   73
continue ?(Y/n)y
Enter operation code: *
84 * 50 =   4200
continue ?(y/n)y
Enter operation code: /
26/51=   .509804
continue ? (y/n)y
Enter operation code: &
code of error: 555
error of operation code
continue ? (y/n)n
Ok.
```

上述程序实现两个任意的两位正整数的加、减、乘、除运算，两位整数利用随机函数产生，用户只须输入“+”、“-”、“*”、“/”运算符，即可完成相应的计算工作。

程序中利用 CAUSE ERROR 语句捕获因运算符输入不正确而产生的运行态错误，这里将错误代码定义为555，出错信息定义为“error of operation code”，本例捕获到的错误可在出错处理模块中处理，即显示出用户定义的出错代码和出错信息。

习 题

11—1. 一个初步编好的程序，可能存在哪几类错误？

11—2. 常用的调试技术有哪些？并简述之。

11—3. 什么是运行错误，这类错误将对程序的运行造成什么影响？

11—4. 为什么说 True BASIC 提供的出错处理结构增加了程序的坚固性？

11—5. 选择第九章若干道习题，在程序中加入出错处理结构，使之能处理因读与写磁盘的误操作引起的运行态错误。

11—6. True BASIC 提供了哪几个出错函数，各有什么作用？

11—7. EXIT HANDLER 语句有什么作用，一般用于何处？

11—8. 试说明 CASUSE ERROR 语句的作用。

第十二章　True BASIC 程序上机操作指导

本章将着重讨论 True BASIC 系统命令，包括文件处理命令，程序的编辑、调试命令，以及程序的编译、汇集和运行命令。最后说明在汉字操作系统状态下如何使用 True BASIC。

下面首先介绍 True BASIC 的运行环境，以及它的全屏幕编辑功能。

第一节　True BASIC 的运行环境

任何一种程序设计语言都有其赖以存在的环境，并非能在任何机器上都能正常运行。环境就是语言存在的客观条件和物质基础。True BASIC 的运行环境主要有两方面：硬件环境和软件环境。

一、True BASIC 的硬件环境

(1) 微型机：IBM PC/XT、AST286/386及任何一种兼容机。

(2) 内存：应大于192KB。

(3) 至少应有一个软盘驱动器或一个硬盘驱动器。

(4) 应有一个单色或彩色显示器。

二、True BASIC 的软件环境

PC-DOS 2.00以上版本的操作系统，或者 CCDOS 2.10或 CCDOD 2.13等汉字操作系统。

第二节　全屏幕编辑

程序输入计算机之后，用户即可利用 True BASIC 提供的全屏幕编辑功能对程序进行编辑修改。全屏幕编辑是对程序进行编辑的一种方式，它主要依靠一些功能键来完成对程序的增、删、改等操作。

一、光标的移动

1. 窗口切换(F1、F2**键**)

启动 True BASIC 后，光标处于背景窗口，按 F1键，可控制光标移到编辑窗口，按 F2键又可控制光标移到背景窗口。

2. 光标上下左右移动键(↑、↓、←、→键)

在同一窗口内，使用↑、↓、←、→键可使光标作向上、下、左、右的移动。但其移动只能一次一个字符位置或一行，而不会移到另一窗口内。编辑程序时，一个语句行编辑结束后不能按回车键，应该使用↑或↓键将光标移到其他欲修改行。否则，或者增加一个空行(光标处于行尾，按回车键)，或者将光示所在位置之后的所有字符作为一个新行。

3. 快速移动光标键

以下四个键可使光标在本行内快速移动：

Ctrl+→	光标右移到本行行尾。
Ctrl+←	光标左移到本行行首。
Tab	光标右移一字。
Shift+Tab	光标左移一字

在 True BASIC 初始状态下，编辑窗口占17行，背景窗口占7行。当程序或运行结果超出此范围时，通常使用以下功能键实现光标快速移动或翻页。

Home	光标移至程序或背景窗口的首部。
End	光标移至程序或背景窗口的尾部。
PgUp	向上翻一页。
PgDn	向下翻一页。

二、初级编辑功能键

1. 插入或替换字符(Ins)

按 Ins 键将改变光标的大小，再按一次，光标恢复原状。当光标较大时，键入的字符将替换光标所指的字符，即处于替换状态；当光标较小时，键入的字符将插入到光标所指字符之前，并使其后各字符右移，即处于插入状态。

2. 插入一个空行(Enter)

将光标移至某行行尾，按 Enter(回车)键，则可插入一个空行，光标停留在新行行首，并使其余各行下移。

3. 删除字或行

退格键(Backspace)　按此键可删除光标所在位置左边的一个字符。

删除键(Del)　按 Del 键，将删除光标所在位置的字符。

删除一个字(Ctrl+PgUp 和 Ctrl+Home)　按前一复合键(Ctrl+PgUp)将删除从光标所在位置开始至此字结尾的所有字符。按后一复合键，将删除从该字首字符到光标所在位置的所有字符。

删除一行部分字符(ESC 和 Ctrl+End)　按 ESC 键，删除光标左侧的全部字符，按复合键(Ctrl+Ene)将删除光标右侧的全部字符。

删除一行　将光标移至行标志处，按 Del 键，将删除光标所在行，并使其余各行上移。

4. 行的连接

若要将相邻的两个语句行合并为一行，可将光标移至第二行的行标志处，按退格键，即可将该行移至上一行的末尾。

三、高级编辑功能键

1. 查询字符串(F3)

F3是 FIND 键。当光标处于逻辑窗口顶部时，按 F3键，将在背景窗口出现“FIND:”字样，此时可按如下格式键入要查找的字符串：

```
  FIND:〈字符串〉
或 FIND:"〈字符串〉"
```

若使用前一种格式，表示查找某个完整的字，且大小写字母不分。若使用后一种格式，表示查找某个字的一部分，且与字母的大小写有关。在查找前，通常应按 Home 键，将光标移到程序开始处，以实现对整个程序的搜索。

2. **标记行模块**(F4)

使用 F4键可以标记一个由若干行组成的行模块。其操作方法是：将光标移到要编辑的行模块的首行，按 F4键，这时该行行首出现长方形的标志并闪烁，表示该行已被标记。再将光标移到行模块的尾行，按 F4键。此时，首尾行之间的所有行(包括首尾行)均被加上长方形标志，即成为一个被标记的行模块。此后，可以对其整体进行复制、移动等各种操作。

3. **复制行模块**(F5)

当程序中需要复制行模块时，应首先将欲复制的行模块打标记，然后将光标移至需要复制该模块的位置，按 F5键后，被标记的行模块就复制到光标所在行的下面，同时被标记行停止闪烁。

若在背景窗口按 F5键，将重现最后一次键入的命令。

4. **上下移动行模块**(F6)

F6键的作用是将被标记的行模块移到(而不是复制)光标所在行的下面，操作方法与复制行模块相同。

5. **左右移动行模块(＞键或＜键)**

被标记的行模块可以整体向左或向右移动，操作方法是：将光标移到行模块任一行的行标志上，若按"＞"键，行模块整体向右移动；若按"＜"键，则行模块整体向左移动。

6. **删除行模块**(Del)

将光标移到被标记的行模块任一行的行标志处，按 Del 键，整个行模块将被删除。

7. **恢复行**(F7)

编辑程序时，若删除某行后发现是误操作，这时按 F7键可重新恢复该行。用 Del、ESC、Ctrl＋End、Ctrl＋Home、Ctrl＋PgUp 等键删去的部分，也可用 F7键恢复，但恢复的条件是按 F7键以前，光标必须仍位于当前行。

8. **运行**(F9)

F9是 RUN 键，按此键后，程序将被执行，其结果或者在背景窗口显示，或者在打印机上打印出来。

四、True BASIC 2.03**版新增的编辑功能键**

Shift＋F1(或 Shift＋F2)　在表列上垂直向上(或向下)移动光标。利用这两个功能键可使用户在缩进格式的程序结构层上，上下移动光标。当在一个结构层中上下移动光标时，如果上下邻近行已不是相同缩进，光标将转向另一结构层表列中。

Shift＋F4　建立标笺行，其行标志处出现一个圆圈，以标志正在修改行。

Shift＋F3　寻找标笺行，利用该复合键可以使光标迅速移到标笺处。

在行模块上加"!"　调试程序时，有时需要将某行模块内各语句暂时设置为注释语句，此时，可先用 F4键标记行模块，然后将光标移到行标记处，按"！"键后，整个行模块中各行行首

均添加“!”注释符。使用同样的方法可删去已加“！”的行模块。

F8　设置新点，可替代 Break 命令。

第三节　True BASIC 系统命令

True BASIC 系统命令可用于屏幕设置、文件的处理、程序的编辑和调试，以及程序的编译和汇集。

True BASIC 允许用户按缩写形式输入系统命令，缩写时至少应拼写命令的前三个字母。如 REPLACE 命令可书写为：

REP；REPL；REPLA；REPLAC；REPLACE

一、屏幕设置命令

1. 设置屏幕颜色

如果计算机配置的是彩色显示器，使用下列各键可改变屏幕颜色：

Ctrl+F　改变光标所在窗口的前景颜色。

Ctrl+B　改变光标所在窗口的背景颜色。

Ctrl+E　改变屏幕边缘的颜色。

2. 设置窗口尺寸(SPLIT)

命令格式为：

```
SPLIT 〔n〕
```

系统处于初始状态时，编辑窗口占17行。利用该命令可重新设置编辑窗口的大小，其中 n 取值范围为0～24。例如，执行：

SPLIT 10

后，编辑窗口占10行，其余为背景窗口。

在 True BASIC 2.03版中，SPLIT 命令无参数时，将在两窗口分界处显示功能键的功能提示。

二、文件处理命令

1. 文件目录操作命令

(1) 改变当前目录命令(CD 或 ENTER)

为方便用户对文件的存取操作，True BASIC 2.03版允许使用 CD 命令改变或查询当前工作目录，其使用方法与 DOS 中的 CD 命令相同。例如：

CD　C:\LI

则系统将当前目录改为硬盘上的 LI 子目录。

(2) 显示文件目录命令(FILES)

命令格式：

```
FILES 〔d:〕〔path〕〔〈文件名〉〕〔.〈扩展名〉〕〕
```

功能：若文件名和扩展名省缺，将显示指定磁盘和目录下的所有扩展名为.TRU 和.TRC

的文件目录。命令中使用通配符“＊”和“?”,将显示指定文件的目录。例如:

```
FILES
```

表示显示当前盘当前目录下所有扩展名为.TRC 和.TRC 的文件名。

```
FILES  B:\user
```

表示显示 B 盘子目录 user 下的所有扩展名为.TRU 和.TRC 的文件名。

```
FILES  A:＊.TRC
```

表示仅显示 A 盘当前目录下扩展名为.TRC 的所有文件名。

2. **清内存命令**(NEW)

命令格式:

```
NEW 〔d:〕〔path〕〔〈文件名〉〔.〈扩展名〉〕〕
```

功能:执行该命令将清内存、清编辑窗口,并使光标移到编辑窗口顶部,等待用户输入源程序。若命令中带有盘符、路径和文件名,系统将为以后使用 SAVE、REPLACE 命令提供这些信息。如果内存中的原有程序需要保留,则在执行该命令之前,应将程序事先存入磁盘,若未存入磁盘,系统将显示:

```
Do you want to save this file?
```

若回答 y,则系统将程序自动存入原文件名中;若回答 n,则将舍弃此程序。然后清内存及编辑窗口,并将光标移至编辑窗口顶部。

3. **文件存盘命令**(SAVE)

命令格式:

```
SAVE 〔d:〕〔path〕〈文件名〉〔.〈扩展名〉〕
```

功能:将内存中的程序以指定的文件名存入磁盘。若未指定扩展名,系统将默认为.TRU。

若使用 OLD 命令将一个程序装载到内存并对其编辑之后,通常应使用 REPLACE 命令重新存盘,如果仍使用 SAVE 命令并以原文件名存盘时,系统将显示:

```
file aready exist
Do you want to over write it?
```

如果回答 y,并按回车键,系统则将编辑后的程序以原文件名存盘,并取代原文件中的内容。例如:

```
SAVE  B:myprog
```

表示将程序以 myprog 为文件名(扩展名为.TRU)存入 B 盘。

4. **装载文件命令**(OLD)

命令格式:

```
OLD 〔d:〕〔path〕〈文件名〉〔.〈扩展名〉〕
```

功能:将指定磁盘指定目录中的程序文件调入内存,并在编辑窗口显示该程序。若文件未加扩展名,系统默认为.TRU。若是编译过的编译代码文件,命令中应加扩展名.TRC。例如

```
OLD  B:myprog
```

表示将 B 盘当前目录中的 myprog(扩展名为.TRU)程序文件调入内存。

5. **重写文件命令**(REPLACE)

命令格式：

```
REPLACE 〔d:〕〔path〕〔〈文件名〉〔.〈扩展名〉〕〕
```

该命令用于程序由OLD命令调入内存并经编辑修改后重新存盘。若命令中给出文件名，则系统将编辑修改后的程序以指定文件名重新写到磁盘原位置上；若省略文件名，则以取出该程序时所用的文件名存盘。例如：

```
REP
```

表示将编辑修改后的程序以取出该程序时所用的文件名存盘。

在True BASIC 2.03版中，如用户修改过当前编辑的程序，但没有用SAVE或REPLACE命令存盘，这时如果打入OLD或NEW或BYE命令，系统将显示：

```
Do you want to save this file?
```

若回答y，则系统将程序自动存入原文件名中；若回答n，则将舍弃此程序，执行所给命令。

6. **删除文件命令**(UNSAVE)

命令格式：

```
UNSAVE 〔d:〕〔path〕〈文件名〉〔.〈扩展名〉〕
```

功能：删除指定盘和目录中的给定文件。文件被删除后，系统将回收文件占用的存储空间。例如：

```
UNSAVE  B:myprog
```

表示删除B盘上名为“myprog.TRU”的文件。

7. **文件改名命令**(RENAME)

命令格式：

```
RENAME 〔〔d:〕〔path〕〈旧文件名〉〔.〈扩展名1〉〕,〕〈新文件名〉〔.〈扩展名2〉〕
```

功能：将指定磁盘和目录中的旧文件名改变为所给的新文件名。若未给出旧文件名，表示改变当前编辑中的文件名，即在窗口分界处显示的旧文件将以指定的新文件名命名。

8. ECHO **命令**

True BASIC 2.03版提供的ECHO命令，其功能是：将程序的运行结果送打印机打印，或者作为文件存入磁盘。该命令有多种使用格式，例如：

ECHO　表示接通打印机，并将屏幕显示结果送打印机打印。

ECHO TO 〔d:〕〔path〕〔〈文件名〉〔.〈扩展名〉〕　表示将程序运行结果以给定文件名存入磁盘和目录中。

ECHO　OFF　表示关闭打印机。

2.03版还允许在一条命令后加入“≫”字符，表示接通ECHO，使命令的输出转到“≫”后面给出的文件中。若“≫”后无文件名，则将结果送打印机打印。例如：

RUN≫　表示将运行结果送打印机打印。

RUN≫ B:address　表示将运行结果存入B盘address文件中。

FILES　B:≫　表示将B盘当前目录中的.TRU和.TRC文件名送打印机打印。

三、程序编辑命令

1. 标记行模块命令(MARK)

MARK 命令用于标记行模块，命令格式为：

```
MARK〈行模块说明〉
```

其中，行模块说明的形式可以是：(1) 一个简单行，如180(表示行号为180)；(2) 一个行号组，如20－90(表示行号20－90)；(3) 一个自定义函数名，如 fact；(4) 一个子程序名，如 swap；(5) 一个图画名，如 star。

True BASIC 允许使用命令为程序语句行设置行号，程序编辑结束后，可再使用命令将语句行号删除。

例如，若标记子程序 swap，可键入命令：

```
MARK swap
```

2. 编辑命令(EDIT)

命令格式：

```
EDIT〔〈行模块说明〉〕
```

功能：清除指定行模块以外的编辑窗口中的其他内容，使以后的编辑操作只能作用于该行模块。如果命令中未指定行模块，而前面已标记了行模块，则编辑操作仍被限制在该模块内，否则，编辑范围为整个程序。例如：

```
EDIT swap
```

表示仅编辑 swap 子程序，即后续编辑命令(如 CHANGE，FIND 等)只对 swap 有效。

3. 替换命令(CHANGE)

该命令通常作用于整个程序，如果使用了 EDIT 命令，则被限制在可编辑的范围内。命令格式为：

```
CHANGE〈字符串1〉,〈字符串2〉
```

功能：把程序中出现的字符串1替换成字符串2。若字符串1未加双引号，则替换完整的字，且与字母的大小写无关；若加双引号，则仅替换某一个字的一部分，且与字母的大小写有关。例如：

```
CHANGE score,newscore
```

表示将程序各行中出现的 score 一律替换为 newscore。

4. 试图替换命令(TRY)

命令格式：

```
TRY〈字符串1〉,〈字符串2〉
```

该命令与 CHANGE 命令的功能基本相同，其区别在于执行该命令时，每次替换之前要询问是否替换，回答 Y 表示替换；回答 N 表示不替换；回答 Q 表示终止替换并退出 TRY 命令状态。

5. 复制命令(COPY)

命令格式：

```
COPY 〔〈行模块1说明〉,〕〈行模块2说明〉
```

功能：将行模块1复制到行模块2的后面。若已用 F4键或 MARK 命令标记了行模块，则行模块1说明可以省缺，否则将显示出错信息。如：

```
COPY 100－160,360
```

表示将行模块(100－160)复制到程序300行之后。

6. **移动命令**(MOVE)

命令格式：

```
MOVE 〔〈行模块1说明〉,〕〈行模块2说明〉
```

功能：把行模块1移动到行模块2的后面。例如：

```
MOVE swap
```

表示将已标记的行模块移到子程序 swap 之后。

7. **删除命令**(DELECT)

命令格式：

```
DELECT 〈行模块说明〉
```

功能：删除命令中指定的行模块。例如：

```
DELECT star
```

表示删除图画 star。

8. **保留命令**(KEEP)

命令格式：

```
KEEP 〈行模块说明〉
```

功能：保留程序中已标记的行模块，删除其余的内容。例如：

```
KEEP indata
```

表示删除子程序 indata 以外的程序行。

9. **文件插入命令**(INCLUDE)

命令格式：

```
INCLUDE 〔d:〕〔path〕〈文件名〉〔.〈扩展名〉〕
```

功能：在程序中插入命令指定的磁盘文件内容，插入的文件内容被置于光标所在行的后面。例如，需要把 B 盘中的 REPORT 文件插入到光标所在行之后，可执行如下命令：

```
INCLUDE  B:REPORT
```

10. **查寻命令**(LOCATE)

命令格式：

```
LOCATE 〈字符串〉(或"〈字符串〉")
```

功能：查寻命令中给定的包含某字或短语的所有程序行，并在屏幕上显示出来。若命令中的字符串未用引号定界，则查寻完整的字，且与字母的大小写无关；若用引号括起来，则查寻某个字的一部分，且与字母大小写有关。例如，要查寻变量 e 所在程序行，可执行命令：

LOCATE e

11. 程序格式化命令（DO FORMAT）

命令格式：

```
DO FORMAT
```

功能：该命令将把程序中的关键字改为大写形式，并按结构层次向右缩进整理 IF、SELECT CASE、循环语句，以及子程序、多行自定义函数、图画等结构，使程序格式清晰。

执行该命令时系统将调用 FORMAT. TRU 程序，因此，输入该命令之前应将系统盘插入驱动器中。

四、程序调试命令

1. 程序的行编辑

调试程序时，除了可以采用全屏幕编辑方式外，还可以用行编辑方式。这一操作方式依靠执行 True BASIC 命令实现对程序的编辑。

执行行编辑时，由于系统需要调用系统盘中的相应程序，所以，需要将系统盘事先插入驱动器中。

（1）设置行号命令（DO NUM）

命令格式：

```
DO NUM
```

功能：执行此命令将为程序各行添加行号。若程序长度小于90行，执行此命令后第一行被设置为100，后续行以10为增量；若程序长度大于90行，则将程序首行设置为1000，后续行仍以10为增量。

（2）更改行号命令（DO RENUM）

命令格式：

```
DO RENUM〈起始行号〉,〈行间增量〉
```

功能：为程序赋以新的行号。例如，执行

DO RENUM 10,10

后，程序首行和行间增量均为10。

（3）删除行号命令（DO UNNUM）

执行该命令将删除程序中的行号。格式为：

```
DO UNNUM
```

2. 设置断点命令（BREAK）

命令格式：

```
BREAK〔〈行模块说明〉〕
```

功能，在行模块的首行设置断点，当程序运行到此处时，首先保留断点之前所有变量的当前值，并把断点处的后续语句显示出来，然后程序停止运行。此时，利用命令语句可以查看断点前各变量的当前值。消除断点的方法是再设置一次断点。

如果命令中未给定行模块，则将断点设置在被标记行模块的首行。如果将断点设置在不可执行的语句行上，系统则自动将断点设置在其后第一个可执行的语句行上。

3. **继续运行命令**(CONTINUE)

程序因设置断点而暂停后，可以利用 CONTINUE 命令由断点处继续运行程序。命令格式为：

```
CONTIINUE
```

如果因执行 STOP 或 END 语句导致程序停止运行，此命令将无效。

4. **跟踪调试命令**(DO TRACE)

该命令用于对当前程序进行跟踪调试。它的具体使用方法参阅第十一章。

5. **语句命令**

为了方便用户调试程序，True BASIC 允许将某些语句作为命令使用。即在背景窗口可直接键入这些语句，并得到结果。这些语句如表12-1和表12-2所示。

表12-1　**任何时候都可以作为命令语句**

ASK	BOX	CLEAR	GET	INPUT	LET	LINE INPUT
PAUSE	PLAT	PLOT	PRINT	SET	SOUND	UNSAVE

表12-2　**运行一程序后下列语句可作为命令**

CALL	CLOSE	DRAW	REASE	MAT	OPEN
READ	RESET	RESTORE	WINDOW	WRITE	

由于运行程序后，程序中所使用的变量仍维持当前值，因此，用户可利用上述语句命令检查程序中的错误。

五、程序的编译、汇集和运行命令

1. **编译命令**(COMPILE)

当程序调试结束没有错误之后，用户可以用编译方式将程序转换成编译代码程序，然后用 SAVE 命令存盘，在磁盘上生成扩展名为.TRC 的编译代码文件。与.TRU 文件相仿，该文件仍可在 True BASIC 状态下用 OLD、RUN 命令装入内存并运行，但其运行速度加快，并可实现源程序的保密处理。编译命令格式为：

```
COMPILE
```

执行该命令时，系统首先显示：

```
No saved copy-still want to compile?
```

若回答 n 时，系统将不再进行编译；若回答 y，系统将对内存中的程序进行编译，同时源程序消失。编译结束后，将在内存中生成编译代码程序，并在编辑窗口中出现：

```
(compile)
```

注意，程序一旦编译便不再保留其源程序，因此，若需要保留时，应在执行编译命令之前将程序存入磁盘。

2. 汇集命令(BIND)

为了能产生在DOS状态下可直接执行的.EXE文件,以及实现True BASIC与其他可执行程序(.EXE)间的连接,True BASIC 2.03版提供了BIND.EXE,BIND.RTP及EXEC.EXE等三个文件。其中BIND.EXE用于将编译过的True BASIC编译代码文件(.TRC)汇集成.EXE文件。

如果用户需要产生.EXE文件,应首先做如下初始化工作:

(1) 将True BASIC源程序编译,生成编译代码文件(.TRC),其中应包括程序中使用的所有库文件。

(2)执行BYE命令,退出True BASIC系统,并返回DOS系统状态。

(3) 将存储有BIND.EXE及BIND.TRP文件的软盘插入A驱动器。

然后在A>提示符后键入汇集命令(BIND)。该命令格式为:

```
BIND 〔d1:〕〔path1〕〈文件名1〉〔.TRC〕,〔d2:〕〔path2〕〔〈文件名2〉.EXE〕
```

功能:对.TRC程序文件进行汇集,并在指定磁盘和目录中生成.EXE文件。如果命令中省缺文件名2,执行此命令后将产生与文件名1同名的.EXE文件。若文件名1后的扩展名(.TRC)缺省,则汇集程序默认为.TRC。例如:

```
A>BIND B:exam,C:TBASIC\myprog.exe
```

表示将B盘上的exam.TRC程序进行汇集,在C盘TBASIC子目录中生成myprog.exe文件。

说明:(1) 汇集.TRC程序时,要有已编译过的主程序,并且允许主程序以及程序中使用的库文件在不同的盘和目录中。

(2) 若主程序中使用库文件,并且程序中已用LIBRARY语句指出库文件的盘符和路径,BIND命令中不用附加任何说明。如果需要附加库文件,可在BIND命令中写入相应的库文件名以及盘符和路径,但必须用逗号与其他文件名分隔。

(3) BIND命令行中可以包含多项内容,各项之间必须用逗号分隔。例如:

```
A〉BIND B:main,l1,l2,C:TBASIC\ex.exe,c:a.RTP
```

表示汇集main.TRC,l1.TRC和l2.TRC文件,并使用C盘a.RTP实时模块来汇集它们,汇集过程结束后,将在C盘子目录TBASIC中生成ex.exe文件。

(4) 汇集过程中,若出现错误,将显示出错信息。下面给出各种错误信息的解释。

Library BIND.RTP: No such file　当前目录中没有BIND.RTP文件,应在命令行中指出盘符和路径,或者在当前盘上加入该文件。

Library LLL: Must be compiled　LLL库文件不是.TRC文件,应在使用BIND命令之前,将主程序和所有库文件进行编译。

Library xxx.RTP:Incorrect version of RTP　以.RTP为扩展名的文件格式不正确,或者设计的实时程序包文件(.RTP)有错误。

Library XXX YYY: No such file　在XXX和YYY之间未加逗号,要求文件名之间用逗号分隔。

3. 运行命令(RUN)

RUN命令与功能键F9等价,用来运行内存中的程序,既可以是源程序(.TRU),也可以是

编译代码程序(.TRC),但不能运行经汇集之后生成的.EXE 程序。

运行源程序时,系统进入解释工作方式,一旦发现语法错误,光标立即停留在相应的位置上,并在背景窗口显示出错信息。消除程序中的所有语法错误后,运行程序时才会得到运行结构。

2.03版本还允许 RUN 命令之后跟“≫”和文件名,将运行结果以指定文件名存盘或送打印机打印。

六、其他命令

1. 进入 True BASIC 命令(HELLO)

在 DOS 状态下键入“HELLO”命令并按回车键,即可把 True BASIC 2.03系统从磁盘装入内存。此时屏幕被分成上下两个窗口,并在两窗口之间的分界线中间设置文件名监视口。屏幕上部为编辑窗口,程序的输入和编辑在此窗口内进行。屏幕下部为背景窗口,“OK”是 True BASIC 提示符,可在其后输入各种命令,程序的运行结果也在该窗口中显示。

当用户编辑未命名程序文件时,监视口内显示 UNtitled,当用户存入或从磁盘调入一个已知文件名的程序时,监视口将显示当前的实际文件名。

2. 退出 True BASIC 状态命令(BYE)

执行 BYE 命令后,True BASIC 将停止运行,并返回到 DOS 系统状态。对2.03版本,如果用户修改过当前程序而没有存盘时,打入 BYE 命令后系统将显示:

```
Do you want to save this file?
```

若回答 y,则自动将程序存入原文件中;若回答 n,则系统将舍弃此程序。然后返回操作系统状态。

3. 列程序清单命令(LIST)

执行该命令将在打印机上打印程序清单或程序中的行模块。命令格式为:

```
LIST [〈行模块说明〉]
```

如果命令中给出行模块说明,或者用 F4键或 MARK 命令标记行模块时,将仅打印被标记的行模块。否则,将打印整个程序。例如,执行

```
LIST swap
```

时,仅打印出子程序 swap 程序清单。

4. 分页打印命令(DO PAGE)

命令格式:

```
DO PAGE,〈页标题〉
```

功能:在打印机上分页打印当前程序,每页左上角输出命令中指定的页标题。

5. 批处理命令(SCRIPT)

True BASIC 允许用户建立批处理文件(.TRU)。该文件中每行是 True BASIC 可执行命令。例如,在编辑窗口输入:

```
OLD myprog
LIST
RUN≫
```

然后以 prog 为文件名存盘，接着执行：

```
SCRIPT prog
```

将顺序执行 prog. TRU 文件中给出的各命令。

SCRIPT 命令中还可以传送参数。如将上述 prog 文件修改为：

```
OLD 〈1〉
LIST
RUN ≫
```

以 newprog 存盘后，以背景窗口可执行：

```
SCRIPT newprog,myprog
```

其执行效果与前一 script 命令等价。

6. 帮助命令(HELP)

执行 HELP 命令将为用户提供 True BASIC 中一些命令的解释。命令格式为：

```
HELP topics(或〈命令名〉)
```

若只键入 HELP 时，屏幕将被清除，并显示帮助信息简介，再按任一键，将返回到清屏前的状态。若键入“HELP topics”，将显示这些命令清单。通常，应在 HELP 后面跟一命令名，此时，命令的拼写不能用缩写形式。

注意，使用该命令之前，应将系统盘插入驱动器中。该命令可以用功能键 F10代替。

第四节 True BASIC 与 DOS 系统以及.EXE 文件间的接口

一、True BASIC 与 DOS 系统的接口

True BASIC 2.03版提供的 SHELL. TRU 程序(包含两个子程序 DOS 和 SHELL)，使用户在 True BASIC 状态下可以直接执行 DOS 系统的命令。使用方法是在调用程序中加入，

```
Library "SHELL. TRU"
```

并利用 CALL 语句进行调用。其格式为：

```
CALL DOS("〈参数〉")
```

其中，参数可以是任一操作系统命令(包括所需参数)。执行该 CALL 语句时，系统首先清屏，然后将给定的 DOS 命令传送给 DOS 命令处理程序，并由系统执行，结束后系统处于等待状态，键入任一字符后，重新返回到 True BASIC 状态。

说明：(1) CALL 语句中的参数可以是空串，即“”，此时，将返回操作系统，并可利用 DOS 中的 EXIT 命令重新返回到 True BASIC 状态。

(2) 调用另一子程序 SHELL 的方法及功能与调用子程序 DOS 基本相同，区别在于不清屏。

(3) 如果使用的 DOS 版本低于3.0时，应在系统配置文件中加入 PATH=drive:，对于3.0以上的 DOS 操作系统，没有此限制。

二、True BASIC 与.EXE 文件间的接口

True BASIC 2.03版提供的 EXEC.TRC，使用户可以在程序中将其作为库文件，实现自 True BASIC 程序调用磁盘上的其他可执行程序(.EXE 文件)。该库程序包括 EXEC 和 EXEC_RETURN 两个子程序。

1. 子程序 EXEC 的调用

调用格式为：

```
    CALL EXEC(name$ ,arg$ )
```

其中，name$ 是扩展名为.EXE 或.COM 的文件名，需要时可前置盘符和路径，arg$ 是一字符串。调用该子程序时，将使 DOS 操作系统终止当前程序，装载命令中给出的文件到内存，并将原程序的参数以字符串形式传递给已装载文件，然后执行该文件。

说明：(1) arg$ 允许传递数据参数到调用程序。它与 DOS 命令行中所输入的参数一样传送到执行程序中。若不传递任何参数，agr$ 应为空串。

(2) 调用 EXEC 子程序之前，系统将关闭所有打开文件。

2. 子程序 EXEC_RETURN 的调用

调用格式为：

```
    CALL   EXEC_RETURN(name$ ,arg$ )
```

其功能与调用 EXEC 子程序相仿，不同之处在于将当前程序保留在内存中，执行完所调用程序后，将恢复并继续执行当前程序。

该调用语句中的参数与 EXEC 子程序的调用相同。但应注意，不能用该调用语句来执行一个“停启和常驻内存”的程序(也称 TSR 程序)，否则将显示出错信息“Can't reclain memory”，并破坏内存中的 TSR 程序。

下面的子程序用于实现 True BASIC 程序与.EXE 或.COM 文件的连接。

```
! Call ≪.exe or .com file≫
SUB exe_file
    LIBRARY “exec.TRU”
    DO
        PRINT “Enter a filepec————————→”
        PRINT “〔d:〕〔path〕filename〔.ext〕”
        LINE INPUT f$
        LET ext$ =ucase$ (f$[len(f$)-3:maxnum])
        IF ext$ <>“.exe” or ext$ <>“.COM” then
           PRINTER “Filepec error !”
           INPUT prompt “Continue ?(Y/N)”:ans$
           IF ucase$ (ans$)=“N” then EXIT SUB
        ELSE
            PRINT “Enter parameter to transmit:”
            LINE INPUT p$
            CALL exec_return(f$ ,p$ )
        END IF
    LOOP
END SUB
```

调用该子程序时，若输入“C:PCTOOLS.EXE”和空串（输入所需传递参数时直接打回车键），则系统将从硬盘读出 PCTOOLS.EXE 并装入内存执行，结束后将恢复执行当前程序。

值得注意的是，调用并执行.EXE 文件时，用户必须考虑机器的内存容量，当内存中无法装入调用程序时，系统将显示“Out of memory”的出错信息。

第五节　在 CCDOS 状态下使用 True BASIC 的方法

True BASIC 不可以在 DOS 操作系统中运行，还可以在 CCDOS 汉字操作系统中运行。此时，程序中的字符量可以是汉字串。

一、设置屏幕模式

机器处于 CCDOS 状态且用“HELLO”命令进入 True BASIC 之后，显示器屏幕将改为西文字符模式。如果需要输入汉字，可以背景窗口键入命令：

```
MODE  HIRES
```

或者按下 Ctrl+F7复合键，屏幕被改为高分辨率模式，此时可输入并显示汉字。若要再改为西文字符模式，可键入 MODE 80命令，或再次按下 Ctrl+F7复合键。

如果需要在运行程序时输出显示汉字，程序中应添加设置屏幕模式语句：

```
SET MODE "HIRES"
```

执行此语句后，屏幕将改为高分辨率模式，即可显示汉字信息。运行程序结束后按任一键，屏幕将返回到西文字符模式。

二、在汉字输入方式下程序的编辑方法

在系统进入汉字输入状态后，所有的 True BASIC 编辑命令仍有效，但屏幕显示稍有变化。

（1）屏幕两个窗口可显示的行数减少，用户可用 SPLIT 命令重新设置窗口大小。

（2）每个程序行的行标志由一个空格取代。如果需要删除某程序行，应将光标移至该行最左端，并按下 Del 键即可。若要调试某程序模块，通常不用 F4键标记行模块，而改用 MARK 或 EDIT 命令。

（3）当光标位于程序行最左端时，按回车键仍可插入一个空行。但此行中仍保留原来的内容，实际上内存中并不存在此行内容（即为空行），因此，可以在此行中直接输入新的内容。若将光标移至上一行末尾，并按回车键，可插入一个空行且不保留任何信息。

（4）使用 Del 键删除一行中的部分内容时，行末尾的一个汉字或字符会重复显示，用户可以不予理采这些字符。

（5）用户应根据机器的不同配置来设置屏幕模式。若显示器为 EGA 型，则“HIRES”应改为“EGAHIRES”。实际上，True BASIC 2.03版最适用于增强彩色显示器及 EGA 卡，此时，汉字编辑状态与西文编辑状态完全一致，并且汉字系统与图形完全兼容。

附录 A　IBM-PC 字符与 ASCII 代码对照表

True BASIC 允许在一个字符串或注释中使用机器能识别的任意字符。在本附录里，将说明 IBM-PC 的字符集以及它们的使用方法。

A.1　显示符号集

下面列出了 IBM-PC 机上的所有字符。其中第一个至第128个字符是标准的 ASCII 字符。这些字符在任何种类的机器上几乎是相同的。IBM-PC 机也是如此。但是，也有象0至31这些特殊符号，它们不可以在屏幕上显示或在打印机上打印出来。

表中第129至256个字符是 IBM-PC 机上特有的字符。若在程序中编写了这些字符的显示和打印，那么一旦换了其他类型的机器，可能产生不同的字符。此外，这些字符在“图形”和“高分辨率”模式时可能不显示。

表 A.1

十进制码值	字　符	十六进制码值	十进制码值	字　符	十六进制码值
000	(null)	00	023	↨	17
001	☺	01	024	↑	18
002	☻	02	025	↓	19
003	♥	03	026	→	1A
004	♦	04	027	←	1B
005	♣	05	028	▣	1C
006	♠	06	029	↔	1D
007	●	07	030	▲	1E
008	◘	08	031	▼	1F
009	○	09	032	(space)	20
010	◙	0A	033	!	21
011	♂	0B	034	"	22
012	♀	0C	035	#	23
013	♪	0D	036	$	24
014	♫	0E	037	%	25
015	☼	0F	038	&	26
016	►	10	039	'	27
017	◄	11	040	(	28
018	↕	12	041	)	29
019	‼	13	042	*	2A
020	¶	14	043	+	2B
021	§	15	044	,	2C
022	▬	16	045	-	2D

十进制码值	字　符	十六进制码值	十进制码值	字　符	十六进制码值
046	.	2E	087	W	57
047	/	2F	088	X	58
048	0	30	089	Y	59
049	1	31	090	Z	5A
050	2	32	091	[	5B
051	3	33	092	\	5C
052	4	34	093	]	5D
053	5	35	094	∧	5E
054	6	36	095	—	5F
055	7	37	096	`	60
056	8	38	097	a	61
057	9	39	098	b	62
058	:	3A	099	c	63
059	;	3B	100	d	64
060	<	3C	101	e	65
061	=	3D	102	f	66
062	>	3E	103	g	67
063	?	3F	104	h	68
064	@	40	105	i	69
065	A	41	106	j	6A
066	B	42	107	k	6B
067	C	43	108	l	6C
068	D	44	109	m	6D
069	E	45	110	n	6E
070	F	46	111	o	6F
071	G	47	112	p	70
072	H	48	113	q	71
073	I	49	114	r	72
074	J	4A	115	s	73
075	K	4B	116	t	74
076	L	4C	117	u	75
077	M	4D	118	v	76
078	N	4E	119	w	77
079	O	4F	120	x	78
080	P	50	121	y	79
081	Q	51	122	z	7A
082	R	52	123	{	7B
083	S	53	124	¦	7C
084	T	54	125	}	7D
085	U	55	126	~	7E
086	V	56	127	△	7F

十进制码值	字　符	十六进制码值	十进制码值	字　符	十六进制码值
128	ç	80	169	⌐	A9
129	ü	81	170	¬	AA
130	é	82	171	½	AB
131	â	83	172	¼	AC
132	ä	84	173	¡	AD
133	à	85	174	«	AE
134	ȧ	86	175	»	AF
135	Ç	87			*(cont.)*
136	ê	88	176	░	B0
137	ë	89	177	▒	B1
138	è	8A	178	▓	B2
139	ï	8B	179	│	B3
140	î	8C	180	┤	B4
141	ì	8D	181	╡	B5
142	Ä	8E	182	╢	B6
143	Ȧ	8F	183	╖	B7
144	É	90	184	╕	B8
145	æ	91	185	╣	B9
146	Æ	92	186	║	BA
147	ô	93	187	╗	BB
148	ö	94	188	╝	BC
149	ò	95	189	╜	BD
150	û	96	190	╛	BE
151	ù	97	191	┐	BF
152	ÿ	98	192	└	C0
153	Ö	99	193	┴	C1
154	Ü	9A	194	┬	C2
155	¢	9B	195	├	C3
156	£	9C	196	─	C4
157	¥	9D	197	┼	C5
158	R̈	9E	198	╞	C6
159	ƒ	9F	199	╟	C7
160	á	A0	200	╚	C8
161	í	A1	201	╔	C9
162	ó	A2	202	╩	CA
163	ú	A3	203	╦	CB
164	ñ	A4	204	╠	CC
165	Ñ	A5	205	═	CD
166	a̲	A6	206	╬	CE
167	o̲	A7	207	╧	CF
168	¿	A8	208	╨	D0

十进制码值	字　符	十六进制码值	十进制码值	字　符	十六进制码值
209	╤	D1	233	θ	E9
210	╥	D2	234	Ω	EA
211	╙	D3	235	δ	EB
212	╘	D4	236	∞	EC
213	╒	D5	237	φ	ED
214	╓	D6	238	ϵ	EE
215	╫	D7	239	∩	EF
216	╪	D8	240	≡	F0
217	┘	D9	241	±	F1
218	┌	DA	242	⩾	F2
219	█	DB	243	⩽	F3
220	▄	DC	244	⌠	F4
221	▌	DD	245	⌡	F5
222	▐	DE	246	÷	F6
223	▀	DF	247	≈	F7
224	α	E0	248	°	F8
225	β	E1	249	•	F9
226	Γ	E2	250	·	FA
227	π	E3	251	√	FB
228	Σ	E4	252	n	FC
229	σ	E5	253	2	FD
230	μ	E6	254	■	FE
231	τ	E7	255	(blank)	FF
232	Φ	E8			

A.2　获得键的符号集

表 A.2是 IBM-PC 机键盘的样图及各种可能的击键所产生的代码。这些码可由 GET KEY 语句得到。

注意，若想把码0至255中的任意数值所对应的字符显示出来，可以用 Alt 键和右手边的键盘上的数字键来实现。操作方法是在按住 Alt 键的同时按数字键。然后放开 Alt 键。这时相应的字符就出现在屏幕上。这里只允许使用数字键，而键盘上的最上一行的数字键是不起作用的。

表中，每个键下面的四行数字依次表示：常用码、上档键、控制码和交替码。表中在某些键下面使用了下面几种符号：

— 表示没有字符；

· 表示暂停输出；

↑ 表示靠不住(255,256)；

≠ 表示屏幕打印；

§ 表示使用 Alt 键与数字键，产生0～255之间的任何字符。

表 A. 2

F1	F2	ESC	| 1	@ 2	# 3	$ 4	% 5	ˆ 6	& 7	. 8	(9	) 0	+ -	+ =	←	Num Lock	Scroll Lock		
315 340 350 360	316 341 351 361	27 27 27 —	49 33 — 376	50 64 259 377	51 35 — 378	52 36 — 369	53 37 — 380	54 94 30 381	55 39 — 382	56 42 — 383	57 40 — 384	48 41 — 385	45 95 31 386	61 43 — 387	8 8 127 —	— — — —	— — STOP —		
F3	F4	← →	Q	W	E	R	T	Y	U	I	O	P	| [	| /]	| ←	7 Home	8 ↑	9 PgUp	—
317 342 352 362	318 343 353 363	9 271 — —	113 81 17 255	119 87 23 273	101 69 5 274	114 82 18 256 ↑	116 84 20 276	121 89 25 277	117 85 21 278	105 73 9 279	111 79 15 280	112 80 — 281	91 123 27 —	93 125 29 —		327 55 375 §	328 56 — §	329 57 388 §	45 45 — —
F5	F6	Ctrl	A	S	D	F	G	H	J	K	L	: ;	″ /	- \		4 —	5	6 —	+
319 344 354 364	320 345 355 365	— — — —	97 65 286	115 83 1 287	100 68 288	102 70 4 289	103 71 6 290	104 72 7 291	106 74 8 292	107 75 10 293	100 76 11 294	59 58 12 —	39 34 — —	96 126 — —	13 13 — —	331 52 10 §	— 53 371 §	333 54 — §	
F7	F8	⇧	| \	Z	X	C	V	B	N	M	<	>	? /	⇧	PnSc	1 End	2 ↓	3 PgDn	
321 346 356 366	322 347 357 367	— — — —	92 124 28 —	122 90 26 300	120 88 24 301	99 67 3 302	118 86 22 303	98 66 2 304	110 78 14 305	109 77 13 306	44 60 — —	46 62 — —	47 63 — —	— — — —	42 ≠ — —	335 49 373 §	336 50 — §	337 51 374 §	
F9	F10	Alt												Caps Lock		0 Ins		Del	
323 348 358 368	324 349 359 369	— — — —	32 32 32 32											— — — —		338 48 — §		339 46 — —	43 43 — —

附录B　出错信息注释

True BASIC 处理的错误信息一般情况下可分为下述三类：

* 命令错

* 编译态出错

* 运行态出错

当给 True BASIC 一个不正确的命令时，就会发生命令错。当 True BASIC 编译源程序时，所出现的错误称为编译态出错。当程序建立完毕开始运行时，所捕捉到的任何错误均称为运行态出错（见第十一章）。在此附录中，将给出 True BASIC 的运行态中两个出错信息表。第一个表（B. 1）是按字母顺序排列的各种"出错信息解释；第二个表（B. 2）是按出错信息的代码顺序排列的"出错代码及出错信息"。

B. 1　出错信息解释

Argument types don't match.

当调用带参子程序时，可能此错出现在过早地使用这个子程序。也就是说有关的子程序还没有定义；或者是使用不同的参数来调用同一个子程序。即：有可能调用的参数个数不对，或类型不一致。此时应核对一下子程序中参数的定义及调用时所使用的相应值（变元）。

Bad FIND item; try using quotes.

当试图寻找一个包含逗号或双引号的符号串时，被查找的串必须用双引号括住。（此规则在 INPUT 语句和 DATA 语句中也具有同等效力）。

Badly formed USING tring (8201)

在 USING $ 函数或 PRT　USING 语句中所使用的格式串有错。至少有一项是不符合 True BASIC 规则的。

Badly formed input line. (8105)

在回答 INPUT 语句时格式不正确。（它不是用于读磁盘就是用于读键盘。）最常见的错误是丢失了双引号。

Can't change that key

True BASIC 不允许改变一般显示字符的键的含义。

Can't continue.

通常可以用 CONTINUE 命令来继续运行一暂停的程序。然而 True BASIC 有时是拒绝此动作的。有这样一些可能的原因：当再次运行之前修改了此程序；当程序是由于出错而停止的运行；在使用 CONTINUE 命令之前用过 DO 命令；虽然设置了运行的断点，但仍然是错误引起的停止运行。这在调试程序时经常碰到，上述一切均为不能再继续运行此程序了。

Can't Copt region itself.

True Basic 的编辑程序不允许复制自身的某些程序行。例如，把某个子程序再复制到那个子程序的本身之中。如果非要这样做，应先把此子程序复制到别处。改名后，再复制到原来的子

程序中来。然后再做其它整理。

Can't do graphics on this Computer. (11000)

你的计算机不能画图是由于它没有图形卡，或者你没有使用相应的 SET WINDOW、PLOT、DRAW、BOX、FLOOD、GET POINT 或 GET MOUSE 语句。

Can't edit compiled program.

当你的程序被编译以后就不能再被修改了。对于用 COMPILE 命令编译过的程序，只能用 RUN、SAVE 或 REPLACE 命令来处理。如果你还想改动，应事先留有付本、对付本可再次进行编辑。

Can't erase file not opened as OUTIN. (7301)

如果文件没有用 ACCESS OUTIN 打开，对它是不能使用 ERASE 语句的。

Can't find library name.

在使用 LIBRARY 语句时应有一文件名。但此文件当前不存在。你应检查一下记录此文件的磁盘是否插入计算机上了。

Can't help with that. Try HELP TOPICS.

当你求助回答时，不用 HELP 命令就得用 HELP 键。否则 True BASIC 是不能对你所提的问题给以帮助的。当你用 HELP TOPICS 时，既要拼写正确(不能缩写)，还要 True BASIC 盘在机器中。键入后 True BASIC 才会对此问题给出完整的解释。

Can't input from OUTPUT file. (7303)

用 ACCESS OUTPUT 打开的文件是不能用来读取数据的。如果你非要读，则应把你的 OPEN 语句选择项改或 ACCESS OUTIN.

Can't invert ingular matrix. (3009)

当使用矩阵函数 Inv 时，若打算对奇异矩阵求逆是不行的。

Can't output to INPUT file. (7302)

用 ACCESS INPUT 打开的文件，是不能向其写出数据的。如果非要写，则应把 OPEN 语句选择项改成 ACCESS OUTIN。

Can't PRINT to middle of text file. (7350)

正文文件的数据不得重写。要写只能用 RESET 语句把文件指针移到文件末尾再写，或者删除此文件内容后再写。

Can't SET RECSIZE on non-empty record file. (7250)

当某一记录文件被建立以后，它就具有固定的记录长度。在改写或重读此记录文件时不能改变文件记录长度。

Can't SET WINDOW in picture. (11004)

在图画中不能重新设置窗口或屏幕坐标，所以图画中不能使用 OPEN SCREEN 或 SET WINDOW 语句。

Can't use ANGLE (0,0). (3008)

函数 ANGLE(0,0)中的参数至少要有一个不是零。

Can't use this statement here.

在使用 True BASIC 控制结构时，不能写错位置。例如，SELECT CASE 结构中的 CASE 写错了位置，或丢失了；IF-THEN 结构中用错了 ELSE IF；SELECT CASE 行上多加了不必要

的语句，等等。

Can't use ＃ 0 here.（7002）

对＃0通道是不能使用 OPEN 或 CLOSE 语句的，因为＃0通道总是打开的。

Channel is already open.（7003）

当你试图打开某一通道时，它实际上已经打开了。在你的程序中应检查一下这一通道使用情况。所以要记住：当你不使用某一通道时，应及时关闭它。

Channel isn't a window.（11005）

你在打开窗口，关闭窗口或改换当前窗口的语句中所用的通道号可能是文件的，而不是某个窗口的。请核对一下这类语句中所用的通道号是否正确。

Channel isn't open.（7004）

当你用某个通道来存取文件或窗口时，相应的通道必须是已打开的。在使用通道时，必须用 OPEN 语句打开它。核对一下所给定的通道号是否正确，它是否真的打开了；是否用 CLOSE 语句顺便关上了此通道。

Channel number must be 1 to 1000.（7001）

通道号的数值必须选取在1到1000范围内，不能在此范围之外，请检查或修改所用的通道号。

（Compiled program）.

这是一个提示显示。它出现在你给出 COMPILE 命令之后。编译程序调出来时，在编辑窗口上显示出来。由于编译程序已把源程序代码整理成内部形式，故不能再看到编译前的程序正文，也别打算再修改或编译它了。否则你就应先留有复本。

Constant too large：constant

对你的计算机来说，被显示的数字值太大而无法处理。可用 PRINT MAXNUM 显示你的计算机所能表示的最大数值。此时你应考虑修改有关常数。

Cursor set out of bounds.（8601）

SET CURSOR 语句给出了光标行和列的位置。它们表示光标在当前窗口里应处的位置，每个窗口有它自己的光标坐标。窗口左上角位置总是行为1，列为1。所以你要弄清所有光标是否越界，只需查清窗口右下角的光标即可，这可用 ASK MAX CURSOR 语句来找其行、列数值。

Data item isn't a number.（8101）

你企图从记录文件中读取一个数，但此时文件记录项是一个字符串。故不能向数字变量中赋值。可能是定位有问题。

Data item isn't a string.（8104）

你企图从记录文件中读取一个字符串，但此时文件的记录项确是一个数。故不能向字符串变量中赋值。有可能是定位错。

DET needs a square matrix.（6002）

Det 函数仅用于方阵，这是受数学定义所限。

Disk full.（9006）

当你把文件逐个向磁盘写入时，所占的磁盘空间就越来越大。直至不能再装，即满了。这种错一般由操作系统直接给出。

Diskette removed, or wrong diskette. (9005)

你已经打开了某个文件,True BASIC 将要使用它,可是你却把磁盘取出换上了其他盘,所以出错。已经正在使用的磁盘不可从驱动器中取出!

Division by zero. (3001)

有一表达式是打算用零来除某一数值。如果你想用它来代替最大允许值,并且还继续运行,那就得使用 WHEN 语句来括住此表达式,见下例:

```
When error in
  let x=(1+2+3)/0
use
  let x=Maxnum
end when
```

Maxnum 是 True BASIC 的一个内部函数,在计算机上它可以给出最大允许值。

Doesn't belong here.

在程序中,光标所处的位置不正确。有些语句在使用中会碰到这种情况。应把光标移到相应菜单的适当语句上。调整光标到正确位置并继续。

Ending doesn't macth beginning.

在使用某些控制结构,如 FOR-NEXT 或 IF-THEN-ELSE 等的时候,必须开头与结尾相互呼应。通常容易丢掉了 END 语句;或者是在循环语句中头、尾的循环变量不相呼应,应仔细地阅读这些程序,找出不匹配的地方。

Error in PLAY string. (4501)

在 PLAY 语句中给定的字符串不符合 True BASIC 文法。

Expected "thing".

当前光标所指向的位置上不是 True BASIC 允许的字或标点符号,而是一些其他的东西。这一信息对于你重新修补某个语句是有益的。

Expected ralational operator.

当前光标所指的位置上,必须放上某关系运算符。例如"="或"<"等等。最终,程序于此必须写出某种比较运算。(True BASIC 不允许使用"IF a then…"或"IF a <>0 then…"等形式的判断条件)。

File already exists. (9004)

当建立某一新文件时,实际上它已存在。应核对一下有关的文件名。如果你打算重写某个已存在的文件,那么在 OPEN 语句中把 CREATE NEW 应改写成 CREATE NEW OLD。SAVE 命令不能用而要用 REPLACE 命令。

File is read or write protected. (9001)

当试图读、写某文件时,操作系统对此文件封读或封写。True BASIC 是不会处理封读或封写的。所以应该退出 True BASIC 解决操作系统对此文件所做的封读或封写动作。

File pointer out of bounds. (7252)

你打算用 RESET 或 SET POINTER 或 SET RECORD 语句来改变文件的指针。但是,现在所给定的文件指针不是比1小,就是超出了文件结束符以外。可用 ASK FILESIZE 语句查出文件有多长。

File's record size doesn't match OPEN RECSIZE. (7103)

每个记录文件都由它的 RECSIZE 建立相应的记录长度。当被打开文件的 RECSIZE 与你给定 OPEN 语句中的 RECSIZE 不一致时就出现此问题。如何才能正确地打开此文件呢?简单的方法是使用文件的记录长度。这时你可以把 OPEN 语句中的 RECSIZE 部分删掉。若还想在 OPEN 语句中使用 RECSIZE,那么你可以先执行一小段检测程序,查出文件的 RECSIZE 赋值后再打开它。这样就会用到 ASK RECSIZE 语句。

IDN must make a square matrix. (6004)

单位矩阵必须是方阵。另外当使用 IND(x,y)函数时,必须 x=y。

Illegal array bounds for name in routine.

在数组的定义中,上界小于下界是非法的,核对一下相应 DIM 语句所设置的数组。(实际上,True BASIC 允许下界比上界大1。但比1大是不行的。这时所定义的数组是不能含有元素的。)

Illegal channel.

在使用 SET 或 ASK 语句时,通常应给出所需的通道号,这里是由于没有给出而非法,应该查出加上。但有时在 SET 或 ASK 语句中不允许有通道号,可是你却加上了,应查出删掉。

Illegal data.

这是你的 DATA 语句书写不当。逗号只能放在两个数据之间,而不能放在该语句的最后面。所有双引号必须成对书写。

Illegal exit.

书写 EXIT DO 语句是为了退出某个 DO 循环;书写 EXIT FOR 语句是为了退出 FOR 循环,但你不想标明是何种循环时可用 EXIT FRED。检查一下你的 EXIT 语句在相应结构中是否使用正确,主要应检查 EXIT FRED 语句的后缀。

Illegal expression.

光标当前所指的表达式位置有不符合 True BASIC 规则的地方。应检查一下是否用了双操作符(例如:“1++2”),或在一个数字中加了逗号(例如:“1,000”)。另外,所使用的变量名是否合法,等等。

Illegal keyword.

在此,光标指出关键字用得不正确的位置。例如,在 BOX 语句中是否丢失了 LINES、AREA、ELLIPSE 或 CLEAR 等关键字。检查相应语句,并改正之。

Illegal number.

光标所指位置需要一个数字,但没有给出。还可能你所书写的数字不符合 True BASIC 文法。有时候 True BASIC 要求某数字常数是固定类型,不得用其他类型,而你用了。例如:DIM 和 OPTION BASE 语句中只能用整型常数。

Illegal option.

True BASIC 只提供了两种选择语句:OPTION ANGLE 和 OPTION BASE。

Illegal parameter.

当你使用 SUB、DFF 或 PICTURE 语句时,是为了定义一个子程序。有时在其中会用到参数。当使用了同名参数,或与某简单变量名重名时,会出现此错。

Illegal statement

True BASIC 的每一种语句都必须在开始带有关键字。象 LET、SELECT 等等。核对一下你所拼写的关键字是否正确。有些 BASIC 版本在书写赋值语句时可以省略 LET 关键字。但 True BASIC 是不允许省略的。

Improper ORD string. (4003)

Ord 函数所需要的不是一简单字符构成的串，就是由 ASCII 码给定的合法名字串。允许不连通或加有空格。

Improper NUM string. (4020)

用于描述 NUM 函数的串不能表示为 IEEE8-字节的浮点数。检查一下你建立的是否正确。

INPUT item bigger than RECSIZE. (8302)

此条信息提供给有故障的记录文件。它说明文件中的某些记录要比文件定义的记录长度要大。如果磁盘上包含有且用到这种带变异性的文件，此信息就会出现。或者你没有使用 True BASIC 文件，而是干预了其他不明了的字节文件时，也有可能发生。

INV needs a square matirx. (6003)

矩阵变换被定义的只限于方阵。当你试图用 InV 函数转换某个非方阵时，是不允许的。应检查一下你的矩阵是否是两维，且每维是否都有相同的长度。

LBOUND out of range. (4008)

当你用 Lbound(A. 3)调用时，数组 A 不能是三维以上的。检查一下相应的维数使用情况。

LOG of number <=0. (3004)

算术对数自变量不能取负值。

MARGIN less than zonewidth. (4006)

你的 SET MARGIN 语句中所设置的 margin 比 zonewidth 小是不允许的。应检查一下相应的变量值。

Mismatched array sizes. (6001)

在使用 MAT 语句时需要相同大小的数组，但你却用上了不同长度的数组。例如，矩阵加法就需要两个数组有相同的长度。因为它要一起相加。矩阵乘法就要有更复杂的规则，也应有相应等同长度的数组。

Missing end statement.

整个 True BASIC 程序必须包含有 END 语句。但当前你丢掉了。在合适的位置加上它再试一下。

MOD and REMAINDER can't have 0 as 2nd argument. (3006)

Mod 和 Remainder 不允许零作为它们的第二个参数。因为这是等于用零来除某数。应仔细检查一下你所用的参数。

Must be a function name.

当你书写 DEF 或 FUNCTION 语句时，没有在相应关键字后面跟有适当的函数名。

Must be a number.

几乎在任何地方 True BASIC 都允许或用数字表达式或用简单数值。但有些地方只能用数字值。例如：CASE 就不能用表达式，只能用数字值常数。如果你必须用表达式，那就应重写这个 SELECT CASE 结构，而改成 IF-THEN-ELSE 结构。

Must be a picture name.

DRAW 语句应当用名字，而且是图画名字，不能用其他的名字。检查一下你程序中名字的使用。

Must be a string constant.

在 True BASIC 中允许使用字符中常数的地方，几乎都允许使用字符串表达式，但有些地方只允许使用字符串常数。例如，CASE 测试就不能用串表达式。如果你非要用，只能把 SELECT CASE 结构改成 IF-THEN-ELSE 结构。

Must be a subroutine name.

CALL 语句仅能用于调子程序。所给的串必须是子程序名。检查一下你所使用的名字是否正确。

Must be a Variable.

在应当使用变量的地方，你使用了表达式或子程序名。例如，LET、INPUT 和 GET 语句就一定要用变量。检查一下这些语句你所使用的是否正确，是否与子程序、函数、图画重名。

Must be an array.

在 True BASIC 的程序中，很多地方都必须使用数组变量，而不是一简单变量。例如，MAT 语句的操作对象只能是数组。许多函数，例如 Lbound 和 Size 等，它们的某些参量也只能是数组。因此你应核对一下数组的名字是否拼写正确，是否与 DIM 语句中定义一致。

Must be byte file for READ BYTES. (7351)

在正文文件或记录文件中使用了 READ BYTES 语句，这是不行的！该语句只能适用于字节文件。检查一下你的 OPEN 语句中为打开这一文件所包含的 ORGANIZATION BYTE 部分。

Must be byte file or empty for SET RECSIZE. (7251)

SET RECSIZE 仅能用于字节文件，或空白的记录文件。正文文件不能有记录长度。非空的记录文件有它自己的记录长度。检查一下你所使用的通道是否正确。

Must be record or byte file for SET RECORD. (7202)

SET RECORD 不能用于正文文件。你可以把一正文文件的指针重新设置到文件的最开始或文件的末尾。

Must be record or byte file. (8502)

当你使用 READ 或 WRITE 语句时，不能用于正文文件或一个窗口。它们只能工作在记录文件或字节文件。核对一下你所使用的通道是否有上述情况。对于正文文件及窗口只能使用 INPUT 和 PRINT 语句。

Must be text file. (8501)

当你使用 INPUT 或 PRINT 语句时，不能用于记录文件或字节文件，它们只能工作在正文文件中。记录文件或字节文件应使用 READ 或 WRITE 语句。可用 ASK ORGANIZATION 询问一下文件的类型。再检查你是否使用正确。

Must SET RECSIZE befor WRITE. (8304)

在向一个空记录文件上写入时，没有定义文件的记录长度。在你的第一条 WRITE 语句之前必须执行 SET RECSIZE 语句，或用 OPEN 语句来指定文件记录长度。

Name can't be redefined.

对任何两个不同的内容不可使用同一个名字。譬如，如果变量名定义为 X，那么子程序名就不能是 X。对于重名错 True BASIC 总是判出错。

Nagative number to non integral power. (3002)

当你计算 n 的 X 次幂时，n 为负数且 X 为非整数，其结果在数学上无意义。

No CASE selected. but no CASE ELSE. (10004)

在执行一条 SELECT CASE 语句时，CASE 测试不成功，同时又没有 CASE ELSE 语句块，True BASIC 就显示上述信息。检查一下你的选择表达式是否使用得完整。若加上 CASE ELSE 在测试时可处理各种情况的问题，可试一下。

No get mouse ON THIS COMPUTER. (8700)

你企图用 GET MOUSE 语句，但在你的计算机上并没有加上鼠标定位装置。

No main program.

当前的程序文件中，仅有函数，图画或/和子程序，但就没有主程序。True BASIC 无法执行这样的程序。

No Saved copy-still want to compile?

当你给过 COMPILE 命令之后，True BASIC 将要抛弃你当前程序内容，而以不可见的编译代码记盘。为了提醒你注意，事先做好必要的复制工作，特出示此信息。“n”为退出此命令；“y”为真正开始编译。故在你回答时，应小心检查一下是否留下了，或打算留下付本。

No such color. (11008)

你用 SET COLOR 语句所设置的颜色名 True BASIC 不认识。颜色名可用大写，也可用小写，但不能在名字中使用多余的符号。再按 True BASIC 能确认的颜色名的完整清单，检查一次你的程序。

No such file. (9003)

你要打开的文件实际上不存在。从很多命令中也可能得到这个信息。(例如 OLD)检查一下名字是否用对了；有关的磁盘是否插入到驱动器里了。还可以用 FILES 命令再检查一下当前的磁盘内容。

(True BASIC 也象一些操作系统一样使用了扩展名。例如，所有的 True BASIC 程序都自动加上了“. TRU”的扩展名。这一点也应引起你注意。)

No such function or subroutine.

在某命令中给出了某个函数、子程序或图画的名字，但这些子程序却不存在。例如，你可键入 LIST MYFUNC，而名字也确实拼写正确了，但 MYFUNC 不存在。

No such line numbers.

命令中所给出的行号与实际程序中的行号范围不符。

No USING item for output. (8202)

在使用 Using $ 函数或 PRINT USING 语句时，应有格式描述串。但是这里没有，而仅有的是有关的正文。

Not found

你可以使用 FIND 键来查找某些字或短语，但并没有找到。记住，True BASIC 总是匹配整字。如果你不打算找一整字，象“print”。而是要找“i”，那么必须用双引号括住所要查找的有关内容。例如：

Find:“i”

这样 True BASIC 将在“print”中找出 i 来。还要记住，True BASIC 总是从光标位置开始一直找到文件的末尾，如果所要找的内容在光标的前面，那你只有移动光标后方能找到。

No index out of range, no ELSE given. (1000)

当用了 ON GOTO 或 ON GOSUB 语句时，所指定的分支号出界了。例如，若 x 比1小或比3大，而用“ON X GOSUB 100,120,130”时，就会出现这样的信息。应核对一下相应下标变量的值。

Out of memory. (5000)

这时，问题出在要求提供的存贮器比你当前机器上所支持的还要多。由于 True BASIC 将使用你的计算机所提供的全部存贮器，这只有通过购买存贮器来解决了。另外，你也应做一些必要的在小存贮器环境下的实验。建议如下：

采用小数组，把多维数的数组可化简成一维数组来分块处理。

编译你的源程序，因为 True BASIC 的编译代码要比源程序短小得多。

检查一下所有的子程序，能采用递归调用的尽量写成完美而合法的递归子程序。这也是常用且有效的方法。

你可以试用程序的链接来解决。

如果你是一位高级程序员，可以试用一下 Packb 和 Unpackb 函数作为最后的手段。它可以让你压缩数据而少占内存。

Output item bigger than RECSIZE. (8301)

当你向一记录文件中去写某一数据项时，由于文件建立时设置了固定的记录长度，而使你不能写入超长的数据。可用 ASK RECSIZE 来找出文件记录长度。记住，任何数据项均不能超过记录长度的指定值。每个数字只能是8个字节长。

Overflow. (1000)

当你要计算的数值超出计算机所能处理的范围时就溢出。PRINT MAXNUM 可见到你的计算机所能使用的最大数。如果你打算通过溢出来获得尽可能大的数值，而又不出错。可写成：

```
When error in
    let x=Tan(pi/2)
use
   let x=Maxnum
end when
```

Plese say what lines you mean.

假如你键入了 MOVE 命令，但又没有描述所要移动的行位置。可用 MARK 键来标记行，或者带上移动的行位置说明再键入 MOVE 命令。

Plese try “CHANGE old, new”

当你修改了一短语时，必须同时给出原来的短语和要改的短语。各短语中若包含有逗号或双引号时，都必须用双引号括住其整个内容。

Plese try “Do filename”.

当用 DO 命令时，必须给出文件名。重新键入带文件名的 DO 命令！

Plese try “INCLUDE filename”.

当用 INCLUDE 命令时，必须给出文件名。重新键入带有文件名的 INCLUDE 命令。

Plese try “KEY” or “KEY FROM file” or “KEY TO file”.

KEY 有三种形式。关键字本身是恢复原来键的含义。“KEY TO”和“KEY FROM”让你更改键的定义。

Please try “LOCATE item”.

当你想确定某短语时，必须给出这相应短语。

Please try “OLD filename”.

当你使用 OLD 命令时，必须给出所要装入的文件名。以便 True BASIC 从磁盘中调出此文件。

Please try “SAVE filename” or “REPLACE filename”.

当你对当前文件记盘时，必须给它一个名字。以便在磁盘目录中登记。

Please try “UNSAVE filename”.

当你删除某文件时，也必须给出这个文件名。以便在磁盘目录中去掉此文件。

Please type line numbers as 100 or 100-150.

假如你所给出的命令是 DELETE，且所带的参数应是一个行号或由行号表示的行模块。不带后面这些说明项 True BASIC 就不懂了。DELETE 100是删除第100行。DELETE 100-150是删除100到150行的程序块。

Program stapped.

按 STOP 键(即 CTRL_SOROLL LOCK 两键同时铵)，或执行 STOP 语句，程序就被终止运行。

REPEAT $ count<0. (4010)

你在使用 REPEAT $ (a $,n)函数时，n 不能是小于零的数。检查一下在此你所使用变量的赋值情况。

Reading past end of data. (8001)

在你执行 READ 语句时，所读的 DATA 项比定义给出的 DATA 项要多，造成读出界。解决的办法有两种。其一，是使你读的项与定义的项在个数上应取一致。其二，读之前可用 MORE DATA 测试一下，再决定是否继续读。

Reading past end of file. (8011)

当你从文件中读多于文件所记录的内容时出此错。适合于各种文件。防止的方式有二。其一，明了文件大小，及记录格式。其二，可用 MORE #n 和 END #n 预先测定相应文件是否结束了。

RETURN without GOSUB. (10002)

当执行 RETURN 语句时，由于没有对应的 GOSUB 语句事先执行过，所以出此错。追踪此错有时是很困难的。建议你使用子程序名字调用方法，这样的结构清晰，而且不容易发生类似错误。不相信可试一下。

Screen bounds must be 0 to 1. (11003)

以 OPEN SCREEN 语句设置的屏幕边界必须在0到1的范围之内。不管你的屏幕有多大，其左下角定义为0；右上角定义1。

Screen minimum>=maximum. (11002)

OPEN SCREEN 语句定义了新窗口。用四个数字来表示，它们的顺序是：左、右、下和上。右边界必须大于左边界；上边界必须大于下边界。如果你打算定义工作窗口，就不能使用此语句。而应使用 SET WINDOW 语句。

SIZE index out of range.（4004）

假如你使用的函数是 Size(A,3)，当数组 A 的维数小于3就会出现此错。应检查 DIM 语句定义的数组维数是否正确。

SQR of negative number.（3005）

不能对负数求平方根！

Statement outside of program.

光标当前所指的程序行既不是属于程序的，又不属于外部任何子程序、函数或图画的。是多余的程序行。True BASIC 不允许这样的程序行出现。主程序一定要以 END 语句结尾。比如，有一程序行在 END 语句之后，且又没有说明它是子程序或外部子程序，则出错。

String given instead of number.（8103）

企图用 INPUT 语句输入某个数值，在此又没有给出数值，而给出了字符串，属类型错。如果你是从文件中读数据，应该查一下文件内数据类型与语句中变量类型是否一致。如果是从键盘上读数据，且读的是个字符串而不是欲读的数值，可用 Val 函数将它转换成数值。

String too long.（1051）

所建立的字符串长于32000个字节。

Subscript outut of boounds.（2001）

数组下标越界。可用 Lbound 和 Ubound 查出数组的相应边界，且在屏幕上显示出来。

TAB column not between 1 and margin.（4005）

在 PRINT 语句中使用 Tab 函数时，所给定的参数不是1至当前行边界之内，出此错。可用 ASK MARGIN 语句找出当前的行边界。该对一下你的使用情况。

The BYE command is just “BYE”.

当退出 True BASIC 时，只能键入“BYE”，而不能附加其他任何东西。

The COMPILE command is just “COMPILE”.

使用 COMPILE 命令时，不能附加任何其他东西。

The CONTINUE command is just “CONTINUE”.

使用 CONTINUE 命令时，不能在此命令上附加任何其他东西。

The RUN command is just “RUN”.

运行当前程序时，只能键入“RUN”，不能附加任何其他东西。

This must first appear in a DIM or DECLEAR DEF.

对于任何数组和外部函数来说，必须先定义后使用！

Too few input items.（8002）

在执行 INPUT 语句时，输入的项数少于你在 INPUT 语句中要求的项数。如果输入来自键盘，将要求你继续输入；如果来自文件，就应核对一下文件中的项数与语句中的项数不一致的原因，然后作相应改动。

Too many channels open.（7102）

如果你想打开十个以上的通道是不行的。True BASIC 在同一时刻只能处理十个通道。你

应适当地关闭一些暂时不用的通道。（某些操作系统仅可同时打开小于十个的通道。可查阅一下用户指南，看看你的计算机能打开多少。）

Too many input items.（8003）

在执行 INPUT 语句时，输入的项数多于你在 INPUT 语句中要求的项数。如果输入来自键盘，则忽略所有多余项；如果来自文件，就应核对一下文件中的项数与语句中的项数不一致的原因，然后作相应改动。

Trouble using disk or printer.（9002）

True BASIC 在某个磁盘上或打印机上发现问题。不同的计算机其出错原因也不尽相同。首先应检查一下电源是否接通。无论是打印机还是磁盘驱动器都是如此。另外，打印机的纸是否装好，它是否处于暂停态；驱动器中是否没装磁盘或磁盘是否插反了等等。如果始终找不到错误，应在你的计算机上运行诊断故障的程序来帮你解决。

Type is wrong for name in routine.

当你调用某个子程序时，用的是另一子程序的名字，这时会出现相应参数不匹配的类型错。否则就是你在调用子程序时，所用变元的类型与定义中相应参数的类型不一致。数字参数不能用字符串来调用，反之亦然。子程序不能作为函数使用，反之亦然。

UBOUND out of range.（4009）

假如你使用了 Ubound（A，3），且 A 是一个小于3维的数组就出错了。检查一下你的 DIM 语句定义。再核对一下你所使用的维数。

Undefined routine name in routine.

不能在某个子程序中再来定义子程序、函数或图画。也就是说子程序、函数或图画不能嵌套定义，而只能嵌套使用。

Unknown OPEN option.（7101）

在 OPEN 语句后面的选择部分 ACCESS、CREATE 或 ORGANIZAITION 没有给出。这种错误的出现有二。其一，是你根本没有写上。其二，是你可能拼错或用的缩写词不对。

VAL String ins't a proper number.（4001）

所用的 Val 函数中，给定的串必须是由某一适当的数组成。

What?（please type HELP）

你所键入的命令 True BASIC 不懂。如果你想求助于 True BASIC 来解释有关的使用。应键入 HELP，然后回车。它将会给出简要的注释。此时，True BASIC 盘应插在磁盘驱动器上。

Window minimum＝maximum.（11001）

当你使用某个 SET WINDOW 语句时，所设置的垂直、水平数值、最大值等于了最小值。True BASIC 不允许这样。因为在这样的窗口里，无法显示任何东西。可能问题出在你把设置边界的顺序搞错了。应该是左、右、下、上。

Wrong number of arguments.

在你调用某个子程序、函数或图画时，所给出的调用参数值有错。检查一下子程序相应的定义及你调用的方式。

Wrong number of dimensions.

在你使用数组时，维数给弄错了！核对一下你的 DIM 语句定义及使用的位置上的维数是否一致。如果你是把一个数组传给某个子程序，那就要检查一下子程序的参数。应记住两维数

组在子程序参数表中必须表示成 A(,);三维应为 A(,,)等等。

Wrong type.

所定义的任何量必须与使用的任何量在类型上保持一致,如串变量不能赋数值。反之亦然。记住,字符串进行连接时,应书写“&”而不能使用“+”。

Wrong type of file. (7104)

True BASIC 所建立的各种文件均有它自己的类型标志。以前记录在磁盘上的文件应与你用 OPEN 语句要打开的文件在类型上保持一致。虽然你的 OPEN 语句使用得合法,但由于与打开的文件类型不匹配也是枉然。

You have two routines called name in rautine.

子程序的重名定义为非法。对任何不同的子程序必须给以不同的名字,否则 True BASIC 不知道调用哪一个。这种人为的二义性是容易避免的。核对一下你的子程序名!

Zero to negative power. (3003)

当你求0的 n 次幂时,n 不能小于0。这在数学上称为无意义。True BASIC 也会给出出错信息。

ZONEWIDTH out of range. (4007)

当执行 SET ZONEWIDTH 语句时,所设置的 Zonewidth 比1小或比 margin 大。可用 ASK MARGIN 语句来找出当前的 margin。

B.2 运行态出错代码及出错信息

下面是除键入错以外的所有运行态出错信息清单。它们是由 Extype 函数送回的出错代码。你可以从表中获得出错的简要解释。所有的出错号均与美国国家标准 BASIC 定义的一致。(编者:此表给出的是英文注释,以方便与屏幕上出现的内容相对照。)

代码	解释
1000	Overflow.
1051	String too long.
2001	Subscript out of bounds.
3001	Division by zero.
3002	Negative number to non-integral power.
3003	Zero to negative power.
3004	LOG of number<=0.
3005	SQR of negative number.
3006	MOD and REMAINDER Can't have 0 as 2nd argument.
3008	Can't use ANGLE(0,0).
3009	Can't invert sigular matrix.
4001	VAL string isn't a proper number.
4003	Improper ORD string.
4004	SIZE index out of range.
4005	TAB column not between 1 and margin.
4006	Margin less than zonewidth.

4007	ZONEWIDTH out of range.
4008	LBOUND out of range.
4009	UBOUND out of range.
4010	REPEAT $ count<0.
4020	Improper NUM string.
4501	Error in PLAY string.
5000	Out of memory.
6001	Mismatched array sizes.
6002	DET needs a square matrix.
6003	INV needs a square matrix.
6004	IDN must make a square matrix.
7001	Channl number must be 1 to 1000.
7002	Can't use #0 here.
7003	Channel is already open.
7004	Channel isn't open.
7101	Unknown OPEN option.
7102	Too many channels open.
7103	File's record size doesn't match OPEN RECSIZE.
7104	Wrong type of file.
7202	Must be record or byte file for SET RECORD.
7250	Can't SET RECSIZE on non-empty record file.
7251	Must be byte file or empty for SET RECSIZE.
7252	File pointer out of bounds.
7301	Can't erase file not opened as OUTIN.
7302	Can't ouput to INPUT file.
7303	Can't input from OUTPUT file.
7350	Can't PRINT to middle of text file.
7351	Must be byte file for READ bytes.
8001	Reading past end of data.
8002	Too few input items.
8003	Too many input items.
8011	Reading past end of file.
8101	Data item isn't a number.
8103	String given instead of numer.
8104	Data item isn't a string.
8105	Badly formed input line.
8201	Badly formed USING string.
8202	No USING item for output.
8301	Output item bigger than RECSIZE.
8302	Input item bigger than RECSIZE.
8304	Must SET RECSIZE befor WRITE.
8501	Must be text file.
8502	Must be record or byte file.

8601	Cursor set out of bounds.
8700	NO GET MOUSE on this computer.
9001	File is read or write protected.
9002	Trouble using disk or printer.
9003	No such file.
9004	File already exists.
9005	Diskette remaved or wrong diskette.
9006	Disk full.
9666	Program stopped.
10001	ON index out of range, no ELSE given.
10002	RETURN withoout GOSUB.
10004	NO CASE selected, but no CASE ELSE.
10005	*.
11000	Can't do graphics on this computer.
11001	Window minimum=maximum.
11002	Screen minimum>=maximum.
11003	Screen bounds must be 0 to 1.
11004	Can't SET WINDOW in picture.
11005	Channel isn't a window.
11008	No such color.

* 此表中的10005号出错是针对CHAIN语句的，它描述的是一种特殊的错误，类似编译态的“Illegal statement”错，但它出现在程序中CHAIN及RETURN位置上。

378846

哈尔滨船舶工程学院图书馆
中文藏书

参考文献

〔1〕N. 沃思著，曹德和、刘椿平译：《算法＋数据结构＝程序》，科学出版社，1984

〔2〕D. E. Knuth 著，管纪文、苏运霖译，《计算机程序设计技巧》，第一卷，国防工业出版社，1980.

〔3〕邹海明、余祥宣：《计算机算法基础》，华中工学院出版社，1985.

〔4〕陈景润著：《初等数论》，科学出版社，1978.

〔5〕严蔚敏、吴伟民编著：《数据结构》，清华大学出版社，1992.

〔6〕许卓群、张乃孝、杨冬青、唐世渭：《数据结构》，高等教育出版社，1987.

〔7〕谭浩强、张基温、徐士良：《程序设计与开发技术》，清华大学出版社，1991.

〔8〕刘甲耀、尹桂兰编著：《PAD 编程方法与 C 语言程序设计》，电子工业出版社，1989.

〔9〕John G. Kemeny, Thomas E. Kurtz, "True BASIC", 1985.

〔10〕John G. Kemeny, Thomas E. Kurtz，秦笃烈译，《结构化 True BASIC 程序设计》，北京大学出版社，1990.

〔11〕谭浩强、张基温：《True BASIC 程序设计》，清华大学出版社，1989

〔12〕刘桂芝、王伟立编著：《True BASIC 程序设计教程》，中国科学技术出版社，1989.

〔13〕高永泉主编：《结构化 BASIC 程序设计教程—— True BASIC》，电子工业出版社，1990.

〔14〕秦笃烈、刘学忠等编译：《最新 Turbo BASIC 实用大全》，海洋出版社，1990.

〔15〕吴文虎主编：《信息学（计算机）奥林匹克》，北京大学出版社，1992.

〔16〕郑人杰：《计算机软件测试技术》，清华大学出版社，1992.

〔17〕李世民、潘晓今、杨卫东、刘华红编著：《TRUE BASIC 程序设计教程》，北京工业大学出版社，1990.

Document generated by Anna’s Archive around 2023-2024 as part of the DuXiu collection (https://annas-blog.org/duxiu-exclusive.html).

Images have been losslessly embedded. Information about the original file can be found in PDF attachments. Some stats (more in the PDF attachments):

{
  "filename": "MTAyMDU1Mzkuemlw",
  "filename_decoded": "10205539.zip",
  "filesize": 19548374,
  "md5": "be59e3a383725cf69b5edc311975a9f4",
  "header_md5": "5a5aa3de1b0f474a3d1674b21a025339",
  "sha1": "5cf8ee4b213ffde8c73cedd02cb194e3242c1ec1",
  "sha256": "0830ec92bd392bc17d99ff54e683fe22b55e1c5a49c469a9bc4450e875b0b683",
  "crc32": 705741638,
  "zip_password": "",
  "uncompressed_size": 19883033,
  "pdg_dir_name": "True BASIC \u255c\u00df\u2563\u2563\u2557\u00bb\u2502\u2560\u2568\u2265\u2554\u03a6\u255d\u255e_10205539",
  "pdg_main_pages_found": 288,
  "pdg_main_pages_max": 288,
  "total_pages": 299,
  "total_pixels": 2096666744,
  "pdf_generation_missing_pages": false
}